chromasens
Imaging for Professionals

CS-3D-Api | Manual

CD40083 Version E25

Table of Contents

1 About Chromasens

The name of our company, Chromasens, is a combination of 'chroma' which means color, and 'sens' which stands for sensor technology.

Chromasens designs, develops and produces high-quality and user-friendly products:

- Line scan cameras
- Camera systems
- Camera illumination systems
- Image acquisition systems
- Image processing solutions

Today Chromasens GmbH is experiencing steady growth and is continually penetrating new sales markets around the globe. The company's technologies are used, for example, in products and for applications such as book and document scanners, sorting systems and inspection systems for quality assurance monitoring.

Customers from all over the world from a wide range of industrial sectors have placed their trust in the experience of Chromasens in the field of industrial image processing.

1.1 Contact Information

Chromasens GmbH
Max-Stromeyer-Str. 116
78467 Konstanz
Germany
Phone: +49 (0) 7531 / 876-0
Fax: +49 (0) 7531 / 876-303
Email: info@chromasens.de

1.2 Support

Chromasens GmbH
Max-Stromeyer-Str. 116
78467 Konstanz
Germany
Telefon: +49 (0) 7531 / 876-500
Fax: +49 (0) 7531 / 876-303
Email: support@chromasens.de

Visit our website at http://www.chromasens.de which features detailed information on our company and products.

2 General Information

The CS-3D-Api is a C++ library which provides you with functions to calculate 3D information in a fast and scalable manner. Through the vast availability of CUDA capable gaming video cards, which are relatively inexpensive, we can speed up the computation drastically. If you need higher speed you can simply add another video card to your computer.

As input images, one or two RGB/BGR/GRAY images from a stereo imaging system are required. Additional calibration file has to be provided which includes mainly the external and internal camera parameters.

The CS-3D-Api calculates a rectified RGB/BGR/GRAY-image or/and an 8-/16-bit height map image or/and a point cloud.

The CS-3D-Api is available for both Windows and Linux operating systems.

2.1 Software Requirements

2.1.1 Windows

- System
    - Windows 7, Windows 8.1, Windows 10 x64
- NVidia Driver Version 441.87
- Software protection dongle software
- For examples only:
    - Development Environment
        - Microsoft Visual Studios 2010 or newer
        - MFC
    - OpenCV – Library v2.4
- For the HALCON extension:
    - HALCON 12, 13,18.11 and 19.11 are supported
- For the LabVIEW extension
    - LabVIEW 2012 and 2016 are supported

2.1.2 Linux

- System
    - Linux Ubuntu 16.04.06 LTS x64
- NVidia Driver Version 410.57 or higher
    - **Important:**
      It is required to have both the NVidia drivers and the CUDA library installed.

      For example: Install with `sudo apt install nvidia-410 libcuda1-410`
- For examples only:
    - Development environment (build-essential package)
    - CMake 3.10 or higher
    - OpenCV – Library v2.4

2.2 Hardware Requirements

- CUDA 3.5 capable GPU hardware (see our compatibility list) with at least 1.5 GB RAM

- Quad Core >2,4 GHz
- Ram >8 GB
- Power-supply with enough power for the GPU(s)
- Software protection dongle

2.3 Recommended System

- Software
    - Windows 7 x64 or newer
    - For examples only:
        - Microsoft Visual Studios 2010 or newer
        - MFC
        - OpenCV – Library v2.4

- Hardware
    - NVidia GTX 980 4GB RAM
    - Intel i7 3,2 GHz
    - 16 GB RAM
    - Chromasens 3DPIXA stereo camera system

3 Remarks on CS-3D-API for Linux

Please note that the current Linux version of the CS-3D-Api software is still a “Beta” state.

If you encounter problems with the software, please send us a bug report with the following information:

- The version of the 3D-API you are running
- The log output, on Linux the log will be shown in the console window.
- A description of the system you are running the 3D-API on, including operating system version and the used GPU hardware
- A short description of the issue that occurred

Please send the report to the following address:

support@chromasens.de

The contents of the CS-API software package for Linux are different from the software package for Windows:

Component / Feature	Windows	Linux
CS-3D-Api	Yes	Yes
CS-3D-Api Samples	Yes	Yes (partly)
Sample Data	Yes	Yes
3D-Viewer	Yes	No
Software Manager	Yes	No
Configuration from file	Yes	Yes
Configuration from camera	Yes	No
Calibration Verification	Yes	No
Halcon & LabVIEW extensions	Yes	No

3.1 Known Issues on Linux

There are existing known issues in the software as listed below:

1.) Dongle licensing problems: It can happen that the software is not able to authenticate with the USB dongle because of issues with the dongle licensing server.

Workaround: Restart the dongle licensing server by executing the following command:

```
sudo service netcbios restart
```

2.) Sample data is currently placed inside the installation folder, which is `/opt/Chromasens/3D`. To actually modify the examples it is required to copy the contents of the “examples” folder to a location where the user has write permissions, for example `~/Chromasens/3D/examples`.

3.) The CUDA library needs to be installed manually. This is not an issue but for convenience it would be better to have that step done automatically. Please refer to the installation steps for Linux in chapter 4.2 for more information.

4 Getting Started

4.1 Installation on Windows Platform

1) Execute the cs-3d-setup-vX.Y.exe and install the software to the desired directory.

2) During the installation process the availability of NVidia driver and its version will be checked. The current driver for NVidia GPUs can be downloaded at http://www.nvidia.de/drivers. Besides, the VS2015 redistributables and the dongle software will be installed.

REMARK: The software is protected against unauthorized use. So the software protection dongle always has to be inserted in a USB-Port connected to the system running the CS-3D-Api. Each instance of the 3D-viewer/API requires a license.

4.2 Installation on Linux Platform

As a first step of preparation, please make sure the latest NVidia drivers and the CUDA library are installed. Please refer to the software requirements in chapter 2.1.2. For example, to install the NVidia drivers and CUDA library version 410, execute the following command:

```
sudo apt install nvidia-410 libcuda1-410
```

On Linux the CS-3D software package consists of the following components:

- install_cs3dapi.sh

  The main installation script that takes care of installing both debian packages and configuring the dongle server.

- CS3D-{version}.deb

  Important: Please use the prepared installation script install_cs3dapi.sh for installing the CS-3D-Api software package, as it will perform additional required steps.

  This is a debian software package which contains the CS-3D-Api shared library with corresponding headers as well as sample images and source code.

- netcbios-1.6.4-amd64.deb

  Important: This software package should not be installed manually, please use the install_cs3dapi.sh script.

  This is a debian software package that installs the Marx dongle server that is required to make use of multi license dongles.

- netcbios.service

  This a service configuration script for the dongle server which allows running the server as a service.

To install the CS-3D software package, please execute the following steps:

1.) Download and extract the CS-3D software package.

2.) Open a terminal and change the current directory to the location where the CS-3D software package was extracted to.

3.) The main installation script *install_cs3dapi.sh* handles all necessary installation steps. This script requires additional permissions to allow execution, please execute the following command to grant these permissions:

```
sudo chmod +x install_cs3dapi.sh
```

4.) Start the installation by executing the *install_cs3dapi.sh* script and follow the instructions in the terminal. The script might install additional required 3rd party software like OpenCV libraries and asks for permission to do so:

```
sudo ./install_cs3dapi.sh
```

5.) The package will be placed in the following location and folder structure:

/opt/Chromasens/3D/3dapi:	CS-3D-Api header files
/opt/Chromasens/3D/bin:	Location for pre-built sample executables
/opt/Chromasens/3D/data:	CS-3D-Api data folder
/opt/Chromasens/3D/doc:	CS-3D-Api documentation
/opt/Chromasens/3D/examples:	C++ examples
/opt/Chromasens/3D/lib:	CS-3D-Api shared libraries
/opt/Chromasens/3D/samples_images:	Sample data with configuration and images
/opt/Chromasens/3D/share:	CMake script for finding the API library

4.3 *Sample Images*

To start working with the 3D-Viewer or the API without having the image acquisition running, we provide a few sample-images in “%PUBLIC%\documents\Chromasens\3D\sample images\{camera type}” along with the configuration information. The name for the sample image for the single camera system is AB_XXX.bmp, and for the dual camera system the names are A_XXX.bmp and B_XXX.bmp.

Please remember to load the matching config.ini from the samples folder before loading the source files.

4.4 *Example Programs*

With the CS-3D software example programs written in C++, C# and HALCON scripts are provided. CS3DApiStandalone and CS3DApiPerformance are also available as precompiled binary programs. Both can be found in the {appDir}\bins folder. The given example-parameters are valid if your current work directory is {appDir}\bins and you use the standard install path.

Windows Sample Data Location:

On Windows all sample data including the example programs is located in the following folder:

```
%PUBLIC%\documents\Chromasens\3D\
```

Linux Sample Data Location:

On Linux all sample data is located in the installation folder in

Sample data:	`/opt/Chromasens/3D/sample_images`
Examples:	`/opt/Chromasens/3D/examples`

List of Sample Programs:

- examples\C++\CS3DApiStandalone

- This sample calculates the height map, rectified image and point cloud from given image pair. First the source images are loaded then the result images are retrieved. The required parameters are passed through command-line.
- Syntax:

  ```
  CS3DApiStandalone.exe
  -a [image from camera A]
  -b [image from camera B]
  -c [configfile]
  --dll [path to CS3DApi64.dll/libCS3DApi64.so including the filename]
  [-od [output height map image].png/.tiff ]
  [-or [output rectified image]]
  [-or2 [output second rectified image]]
  [-p3d [output point cloud].cs3d]
  ```

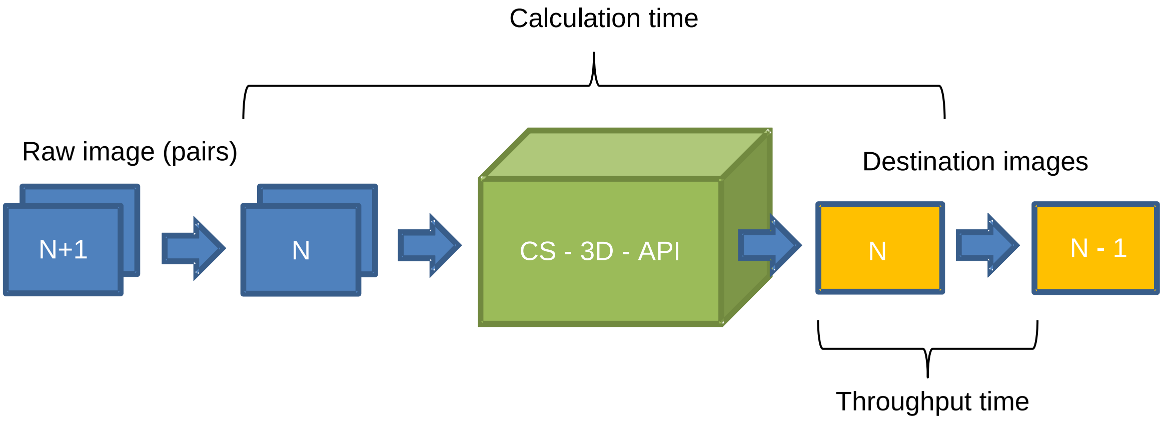
- e.g. for compact camera images:

  ```
  CS3DApiStandalone.exe -a
  "C:\Users\Public\Documents\Chromasens\3D\sample images\single camera
  system\AB_0000.bmp" -c
  "C:\Users\Public\Documents\Chromasens\3D\sample images\single camera
  system\config.ini" --dll "C:\Program Files\Chromasens\3D\dlls\CS3DApi64.dll"
  -od "D:\testHeightMap.tiff"
  ```

- e.g. for dual camera images

  ```
  CS3DApiStandalone.exe -a
  "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera
  system\A_0000.bmp" -b "C:\Users\Public\Documents\Chromasens\3D\sample
  images\two camera system\B_0000.bmp" -c
  "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera
  system\config.ini" --dll "C:\Program Files\Chromasens\3D\dlls\CS3DApi64.dll" -
  od "D:\testHeightMap.tiff" -or "D:\testRectified.bmp"
  ```

- examples\C++\CS3DApiPerformance
  - This sample records the mean calculation time and throughput time per iteration by using parallel processing of loading source image pairs and retrieving result images. The calculation can be done on the whole image or only on a set ROI.
  - Result:
    - “Whole calculation time” is the time that is needed for the calculation of given number of images. Starting when an image is passed to the API stopping when the destination images are ready and copied back.
    - “Throughput time” is the “distance” in second between two consecutive resulting images which the API provides. It can also be interpreted as the speed of the API.



- Syntax:

  CS3DApiPerformance.exe
  -a [image from camera A]
  -b [image from camera B]
  -c [configfile]
  --dll [path to CS3DApi64.dll/libCS3DApi64.so including the filename]
  -i [number of iterations]
  --save [path to save result]
  --useroi [0|1]
  --roix [x-coordinate of a roi]
  --roiy [y-coordinate of a roi]
  --roiw [width of roi]
  --roih [height of roi]
  --usedynamic [0|1]
- e.g. for the whole image calculation without setting roi

  CS3DApiPerformance -a "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\A_0000.bmp" -b "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\B_0000.bmp" -c "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\config.ini" --dll "C:\Program Files\Chromasens\3D\dlls\CS3DApi64.dll" -i 20 --useroi 0

- e.g. for the calculation with a set ROI

  CS3DApiPerformance -a "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\A_0000.bmp" -b "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\B_0000.bmp" -c "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\config.ini" --dll "C:\Program Files\Chromasens\3D\dlls\CS3DApi64.dll" -i 20 --useroi 1 --roix 1200 --roiy 1100 --roiw 500 --roih 300

- examples\C++\CS3DApiSample

  - This sample shows a sequence of image pairs could be processed. The loading of the source image pairs and the retrieving of the resulting images is done in parallel.
  - **Note:** This example is not yet available to Linux installations

- examples\C++\SampleDisplayImages
  - This sample first calculates the height map and rectified image and then displays it. The required parameters are passed through command-line.
  - **Note:** This example is not yet available to Linux installations

- examples\C++\CS3DApiCalibrationVerification
  - This sample performs calibration verification, which checks if the calibration is valid for the current camera status. It takes one (for compact camera) or two (for dual camera) raw images and corresponding config/calibration files as input, and the verification results can be the following three cases:
    - Calibration verification succeeded: Calibration is valid
    - Calibration verification succeeded: Calibration may not be valid any more. Calibration may need to be adjusted, or further test required.
    - Calibration verification failed: Image quality is not acceptable for calibration verification (image is unsharp, or lack of texture in parts of the image), or height range is not set properly, or image is scanned in wrong direction.

    The required parameters are passed through command-line.
  - Only calibration of version 2 or higher can be verified with this function.
  - Syntax:

    CS3DApiCalibrationVerification.exe

    -a [image from camera A]

    -b [image from camera B]

    -c [configfile]

    --dll [path to CS3DApi64.dll including the filename]
  - e.g. for compact camera images:

    CS3DApiCalibrationVerification.exe -a " C:\Users\Public\Documents\Chromasens\3D\sample images\single camera system\AB_0000.bmp" -c "C:\Users\Public\Documents\Chromasens\3D\sample images\single camera system\config.ini" --dll "C:\Program Files\Chromasens\3D\dlls\CS3DApi64.dll"
  - e.g. for dual camera images

    CS3DApiStandalone.exe -a "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\A_0000.bmp" -b "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\B_0000.bmp" -c "C:\Users\Public\Documents\Chromasens\3D\sample images\two camera system\config.ini" --dll "C:\Program Files\Chromasens\3D\dlls\CS3DApi64.dll"
  - **Note:** This example is not yet available to Linux installations

- examples\C#\simpleCSharpExample
  - This sample calculates height and rectified images from a given image pair. Then the result images are saved on the hard-disk.

  - **Note:** This example is not yet available to Linux installations

- examples\C#\threadedCSharpExample
  - This sample shows the throughput of the calculation by using two threads in parallel. The raw image, the configuration file and the iteration number can be set.
  - **Note:** This example is not yet available to Linux installations
- examples\C#\simpleHalconCSharpExample
  - This sample has a similar behavior to simpleCSharpExample. Instead of using the CS-C# wrapper it uses the CS3D HALCON extension.
  - **Note:** This example is not yet available to Linux installations

- examples\HALCON
  - The *.hdev scripts contains an example implementation of the height image / point cloud calculation as well as displaying the results. Also the scripts hold examples for modifying the calculation parameters and additional functions.
  - **Note:** This example is not yet available to Linux installations

4.4.1 Building Sample Programs on Linux

On Linux all sample programs come with a prepared CMake script for configuring the make file that allows building the executable. It is recommended to use CMake version 3.10 or newer for configuring the projects. You can find a pre-built version of CMake either in the Ubuntu repositories or for download on following website: https://cmake.org/download/

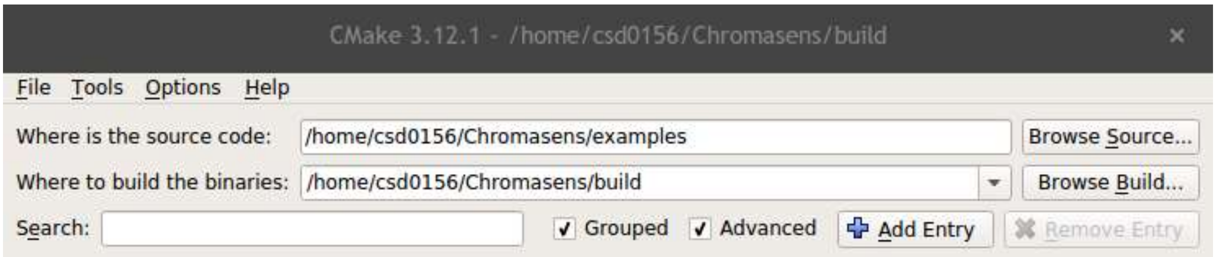
Note: On Linux all sample programs are located in the following folder:
/opt/Chromasens/3D/examples/

This folder is by default not writable - it is therefore recommended to copy the examples folder to another location, for example in the user directory.

Please also note that all sample programs require OpenCV 2.4 or higher to be available on the system. CMake will automatically search for it and report and error if it couldn't find it.

To configure and build an example program please follow the steps below:

1.) Start the CMake graphical user interface called cmake-gui.
2.) In the CMake GUI tool select the source location of the *examples* folder using the "Browse Source…" button.
3.) Next select a location where CMake should place the generated make files for you by clicking the "Browse Build…" button:



CMake 3.12.1 - /home/csd0156/Chromasens/build

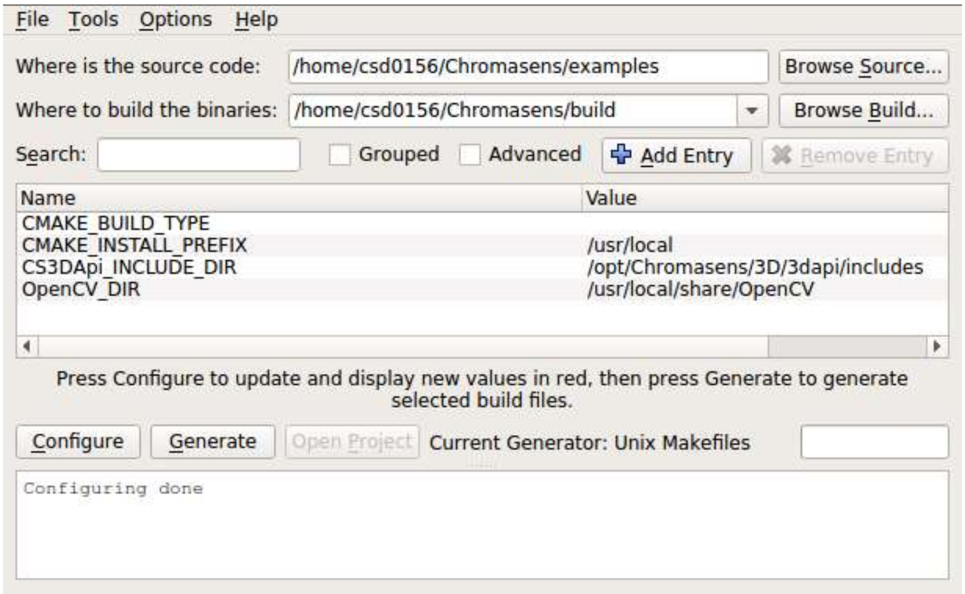
Where is the source code: /home/csd0156/Chromasens/examples

Where to build the binaries: /home/csd0156/Chromasens/build

Search:

☑ Grouped ☑ Advanced

4.) Next click the "Configure" button and allow CMake to create the build folder for you if it doesn't exist yet.
In the upcoming configuration window, select a make file or project generator. CMake is able to either generate a simple "Unix Makefile" or even a project that can be opened in eclipse for further editing.
For this example please select "Unix Makefiles" as the generator.
5.) After pressing configuration two times, your CMake GUI should look like this:



Where is the source code: /home/csd0156/Chromasens/examples

Where to build the binaries: /home/csd0156/Chromasens/build

Search:

Name	Value
CMAKE_BUILD_TYPE	
CMAKE_INSTALL_PREFIX	/usr/local
CS3DApi_INCLUDE_DIR	/opt/Chromasens/3D/3dapi/includes
OpenCV_DIR	/usr/local/share/OpenCV

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

```
Configuring done
```

6.) If no errors occurred, press the “Generate” button to let CMake create the make file.

7.) Next open a terminal and change the current directory to the build folder where CMake created the make files.
For example: `cd /home/csd0156/Chromasens/build`

8.) Next start the build by executing the `make` command.
The built executables will be placed inside a subfolder, for this example it would be
`~/Chromasens/build/C++/CS3DApiPerformance`
and
`~/Chromasens/build/C++/CS3DApiStandalone`

9.) You can now execute the sample programs as described in chapter 4.4 Example Programs.

4.5 Extensions

The following extensions are available for Windows platforms:

- {appDir}\Extensions\Halcon
    - We include an extension for HALCON 12, 13 and 18.11 so HALCON-Users can easily start to use the CS-3D-Api.
    - An example can be found at %PUBLIC%\documents\Chromasens\3D Examples\Halcon
- LabVIEW
    - The “Chromasens 3D Camera Toolkit” is available for download from the LabVIEW Tool Network (LVTN) http://sine.ni.com/nips/cds/view/p/lang/de/nid/212780

4.6 Calculation Speed

The determining factors for the calculation speed are:

4.6.1 RAM / PCI-E Bandwidth of PC

Because the image data has to be copied from the RAM of your PC to the GPUs and back, we recommend mainboards with a high RAM and PCI-E bandwidth.

4.6.2 Calculation Power / Count of GPUs

In principle more GPUs with more CUDA-cores result in faster calculations. A higher clock frequency of the GPU also results in faster calculation.

4.6.3 How to use the API for Highest Speed

The API is designed to process a continuous stream of video images in high speed. Depending on the desired calculation speed, the calculation can be done on 1 to multiple GPUs. To distribute the work to the GPUs, the images are chopped into work units and distributed to them. Therefore a few images are hold in buffers. To get the maximum calculation speed of the API, these buffers have to be full at any time to make good use of the GPUs. There are also output buffers where the results of the work units are combined into images. These buffers should not be full at any time. We recommend using a separate thread for loading images into the API and for getting the result data from the API.

4.7 CS-3D-API Overview

In Figure 1 we give an overview what the API is capable of.

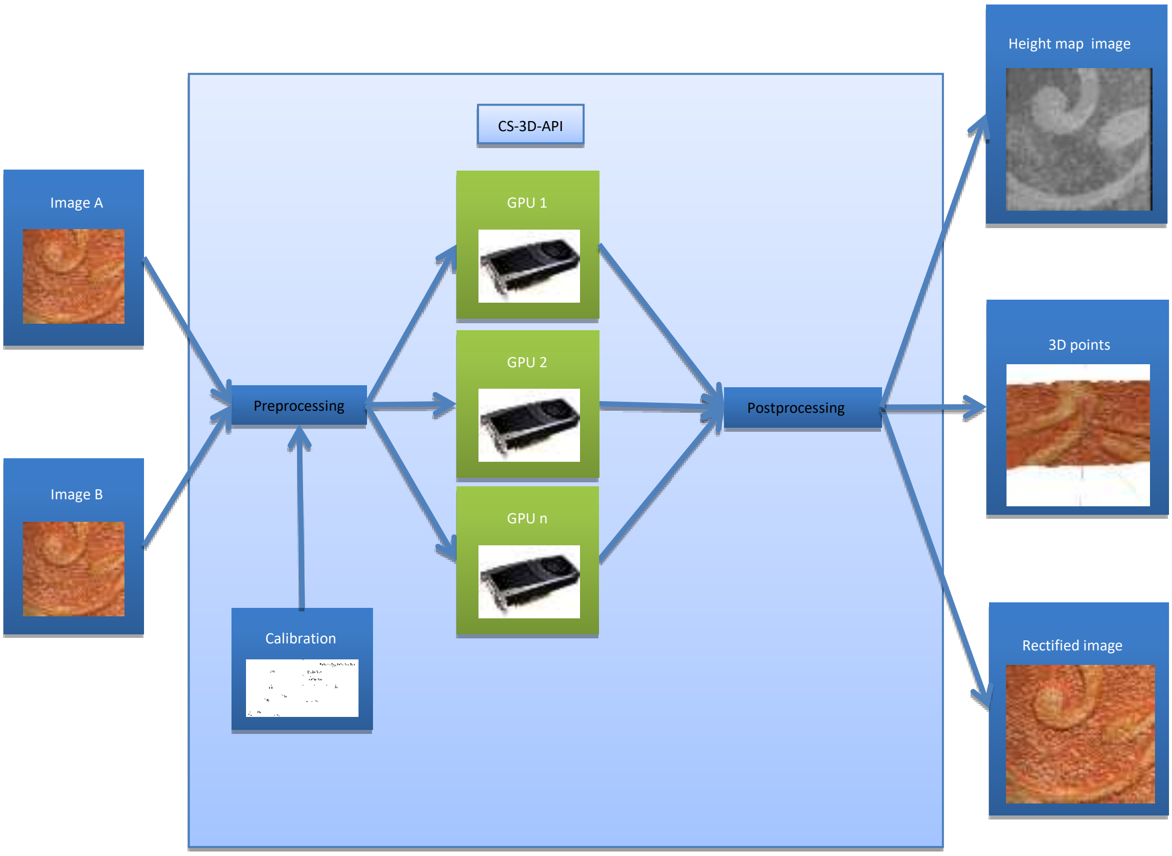


Figure 1: Overview over the CS-3D-Api

5 Api-Wrapper for C#

In the context of software libraries, a wrapper bridges two programming languages so that a library written for one language can be used in another language. For example, the CS-3D-Api is implemented with programming language C++. We implemented a C#-Wrapper “cSharpWrapper.dll” for the API, so that the functions of CS-3D-Api can also be used in C#. You can find that wrapper in the folder {appDir}\bins.

In the following part it will be shortly explained how to use the C#-Wrapper.

Step 1: Create a new / Open a C# project

Step 2: Add cSharpWrapper.dll as reference to the project

> Unfold the project, which was created in step 1. Then right click “Reference” and select “Add reference…”. The Wrapper-File “cSharpWrapper.dll” can be found in the subfolder “bins” of the CS-3D software install path. Add this to the reference. Please make sure you choose the correct DLL-file

Step 3: Create an API object in C#. Then you can access almost every function of the CS-3D-API.

```
public cSharpWrapperApi apiObject = null;
…
apiObject = new cSharpWrapperApi(dllNamePtr, configNamePtr);
…
int ret = apiObject.initialize();
…
ret = apiObject.setSrcImgInfo(…);
```

6 Api - Function List

To avoid complications with other libraries, the Chromasens 3D API uses the CS3D namespace. As the API functions are available for C++ and C#, both variants are listed.

6.1 *Create API Object*

For creating the I3DApi object please do as we do it in the examples and use the helper "accessDllObj" from C:\Program Files\Chromasens\3D\3dapi\helpers If you like to implement the creation on your own. Use the exported "`CS3DApiCreate`" function of the "CS3DApi64.dll"

```
        CS3DApiCreate (I3DApi **ppdst, const char * params, const int majorVersion,
const int minorVersion)
```

Input values:

ppdst - reference to pointer to I3DApi object.

params – can either be empty "" or filename and path to configuration file

majorVersion – must be CS3D_MAJOR_VERSION constant defined in version.h

minorVersion – must be CS3D_MINOR_VERSION constant defined in version.h

Remark	In version v2.3e or older version, the helper files (accessDllObj.h, accessDllObj.cpp, helper.h,helper.cpp) were in the folder {AppDir}\examples\C++\helpers. From v2.4a, the helper files are moved into folder {AppDir}\3dapi\helpers. If your own application is using the path of older helper files, please adjust the path correspondingly.

6.2 *Initialization*

Before starting the calculation, the initialization has to be done to allocate memory for different buffers and start the worker-threads. If the configuration is changed, reinitialization has to be done to activate the changes.

6.2.1 initialize

Variant 1: initialize without parameter

C++	int32_t I3DApi::initialize (void)
C#	int cSharpWrapperApi::initialize (void)

Input values:

None.

Return values:

0 if no error, <0 otherwise.

Description:

Does initialization, must be called after I3DApi creation and before any other functions, except get/load/set/save config functions.

If the configuration parameter “enableDynamicConfiguration” was enabled, the 3D-API will check if an initialization is actually required. Please refer to the description of “enableDynamicConfiguration” in chapter 10.2.2 Control parameters for more information of parameters which don’t require a reinitialization. If such parameters are changed, they will be applied for the next calculation without reinitialization of 3D-API indeed. Please also note that a call to setConfig (6.3.2 setConfig) will also perform a reinitialization of the 3D-API.
If the configuration parameters “enableDynamicConfiguration” was disabled, the 3D-API calculation has to be stopped before initialization.

Variant 2: initialize with parameter

C++	int32_t I3DApi::initialize (config3DApi *config)
C#	Not available

Input values:

config - a config3DApi object to overwrite the initial configuration, loaded from file on I3DApi creation.

Return values:

0 if no error, <0 otherwise

Description:

This function behaves like variant 1, except the given configuration is used instead of the one stored in the API.
Note: Please note the information about the “enableDynamicConfiguration” parameter in variant 1 of initialize.

6.2.2 reinitialize

C++	int32_t I3DApi::reinitialize (void)
C#	int cSharpWrapperApi::reinitialize (void)

Input values:

None.

Return values:

0 if no error, <0 otherwise.

Description:

Does the initialization like I3DApi::initialize (void) (6.2.1) with the current active configuration, except if you set a new configuration with I3DApi::setConfig (6.3.2).

6.2.3 getDemoMode

C++	int32_t I3DApi::getDemoMode (void)
C#	int cSharpWrapperApi::getDemoMode (void)

Input values:

None.

Return values:

1 if demo-mode is enabled, 0 if disabled.

Description:

Returns the state of the demo mode.

6.2.4 setDemoMode

C++	int32_t I3DApi::setDemoMode (int32_t demoMode)
C#	int cSharpWrapperApi::setDemoMode (int demoMode)

Input values:

demoMode – 0 to disable, 1 to enable the demo mode.

Return values:

0 if no error, <0 otherwise.

Description:

Enables or disables the demo mode. After a change, the API has to be reinitialized using functions 6.2.1 or 6.2.2.

6.3 Configuration

There are two configurations present in the CS-3D-Api. The one called active configuration is used by the actual running system. And another configuration called new configuration becomes the active configuration after the call of I3DApi::reinitialize (6.2.2). The two configurations are used to give the user the possibility of choosing the time when the new config becomes active.

The active configuration can only be read by the getActiveConfig or the getConfig function when no new configuration is set. Also it is possible to save the active configuration through saveActiveConfig (6.3.6) or the saveConfig functions, if no new configuration is present.

The new configuration can be set via the setConfig function, read by the getConfig (6.3.1) and written to file via the saveConfig functions (6.3.6).

6.3.1 getConfig

C++	config3DApi* I3DApi::getConfig (void)
C#	bool cSharpWrapperApi::getConfig (void)

Input values:

None

Return values:

C++: NULL on error, otherwise a reference to a config3DApi object.

C#: false on error, true otherwise.

Description:

C++: Returns copy of new configuration object if present, otherwise a copy of the active configuration is returned. The copy of the configuration object has to be freed manually using the freeConfig-function (6.3.10).

C#: Returns a boolean value which indicates if getConfig() runs correctly. To access the result-config and its every individual attribute, you can use the member “config” of the cSharpWrapperApi object. The type of config is CS3DConfig. Its members are numCams, doCalc3DPoints, …

6.3.2 setConfig

Variant 1: setConfig with one parameter.

C++	int32_t I3DApi::setConfig (config3DApi* config)
C#	int cSharpWrapperApi::setConfig (void)

Input values:

C++: config - Configuration object

C#: None

Return values:

<0 if an error occurs, 0 otherwise.

Description:

C++: Sets the given configuration as the new API configuration. After the execution of that method the user has to take care of deleting that object.
Note: Calling setConfig after the API was started is only possible if the configuration flag “enableDynamicConfigration” was set to true. Otherwise it will stop a running API and clear all internal structures.
C#: the config member of the wrapper is set as the new configuration for the API.

Variant 2: setConfig with additional parameter to allow automatic initialization.

C++	int32_t I3DApi::setConfig (config3DApi* config, bool autoInit)
C#	Not available

Input values:

C++:

config - Configuration object

autoInit – If set to false no additional initialization will be done.

Return values:

<0 if an error occurs, 0 otherwise.

Description:

C++: Performs the same steps as variant 1 but additionally allows controlling the initialization step, which is always done in variant 1.

6.3.3 loadConfig

C++	int32_t I3DApi::loadConfig (char* filename)
C#	int cSharpWrapperApi::loadConfig (string filename)

	(deprecated) int cSharpWrapperApi::loadConfig (sbyte* filename)

Input values:

filename – string which holds the filename (including path) of the config-file to be loaded.

Return values:

0 if no error, <0 otherwise.

Description:

Loads a configuration from file and sets this configuration as the new configuration in the API.

6.3.4 saveConfig

Variant 1: saveConfig without parameter

C++	int32_t I3DApi::saveConfig (void)
C#	int cSharpWrapperApi::saveConfig (void)

Input values:

None.

Return values:

0 if no error, <0 otherwise.

Description:

Writes the new configuration back to the original file. If no new configuration is present, the active configuration will be written. Behaves like variant 2 with relativeCalibrationPath = true.

Variant 2: saveConfig with relative file path option

C++	int32_t I3DApi::saveConfig (bool relativeCalibrationPath)
C#	int cSharpWrapperApi::saveConfig (bool relativeCalibrationPath)

Input values:

relativeCalibrationPath – if true only the relative path to the calibration file is stored in the configuration file. If false the absolute path is stored

Return values:

0 if no error, <0 otherwise

Description:

Writes the new configuration back to the original file. If no new configuration is present, the active configuration will be written.

Variant 3: saveConfig with parameter

C++	int32_t I3DApi::saveConfig (char* filename)
C#	int cSharpWrapperApi::saveConfig (string filename)
	(deprecated) int cSharpWrapperApi::saveConfig (sbyte* filename)

Input values:

filename – string which holds the filename (including path) to write the config-file to.

Return values:

0 if no error, <0 otherwise

Description:

Like I3DApi::saveConfig without parameter (6.3.4), but the filename to save the config-file is given. Behaves like variant 4 with relativeCalibrationPath = true.

Variant 4: saveConfig with parameter and relative file path option.

C++	int32_t I3DApi::saveConfig (char* filename, bool relativeCalibrationPath)
C#	int cSharpWrapperApi::saveConfig (string filename, bool relativeCalibrationPath)
	(deprecated) int cSharpWrapperApi::saveConfig (sbyte* filename, bool relativeCalibrationPath)

Input values:

filename – string which holds the filename (including path) to write the config-file to.

relativeCalibrationPath – if true only the relative path to the calibration file is stored in the configuration file. If false the absolute path is stored

Return values:

0 if no error, <0 otherwise

Description:

Like I3DApi::saveConfig without parameter (6.3.4), but the filename to save the config-file is given.

6.3.5 getActiveConfig

C++	config3DApi* I3DApi::getActiveConfig (void)
C#	bool cSharpWrapperApi::getActiveConfig (void)

Input values:

None

Return values:

C++: 0 on error, otherwise a reference to a config3DApi object.

C#: false on error, true otherwise.

Description:

C++: Returns a copy of the active configuration object. The copy of the configuration object has to be freed manually using the freeConfig-function (6.3.10).

C#: Returns a boolean value which indicates if getActiveConfig(void) ran correctly. To call the result-config and its every individual attribute, you can use the member “config” of the cSharpWrapperApi object. The type of config is CS3DConfig. It’s members are numCams, doCalc3DPoints, …

6.3.6 saveActiveConfig

Variant 1: saveActiveConfig without parameter

C++	int32_t I3DApi::saveActiveConfig (void)
C#	int cSharpWrapperApi::saveActiveConfig (void)

Input values:

None.

Return values:

0 if no error, <0 otherwise.

Description:

Writes the active config to the original file.

Variant 2: saveActiveConfig with relative calibration path option

C++	int32_t I3DApi::saveActiveConfig (bool relativeCalibrationPath)
C#	int cSharpWrapperApi::saveActiveConfig (bool relativeCalibrationPath)

Input values:

relativeCalibrationPath – if true only the relative path to the calibration file is stored in the configuration file. If false the absolute path is stored

Return values:

0 if no error, <0 otherwise

Description:

Writes the active config to the original file.

Variant 3: saveActiveConfig with configuration filename as parameter

C++	int32_t I3DApi::saveActiveConfig (char* filename)
C#	int cSharpWrapperApi::saveActiveConfig (string filename)
	(deprecated) int cSharpWrapperApi::saveActiveConfig (sbyte* filename)

Input values:

filename – string which holds the filename (including path) to write the config-file to.

Return values:

0 if no error, <0 otherwise

Description:

Like I3DApi::saveActiveConfig without parameter (6.3.6), but the filename is given.

Variant 4: saveActiveConfig with configuration filename and calibration file path as parameter

C++	int32_t I3DApi::saveActiveConfig (char* filename, bool relativeCalibrationFile)

C#	int cSharpWrapperApi::saveActiveConfig (string filename, bool relativeCalibrationFile)
	(deprecated) int cSharpWrapperApi::saveActiveConfig (sbyte* filename, bool relativeCalibrationFile)

Input values:

filename – string which holds the filename (including path) to write the config-file to.

relativeCalibrationPath – if true only the relative path to the calibration file is stored in the configuration file. If false the absolute path is stored

Return values:

0 if no error, <0 otherwise

Description:

Writes the active config to a given configuration filename. Additionally, the user decides whether the relative calibration file path is used

6.3.7 loadConfigFromCamera

C++	int32_t I3DApi::loadConfigFromCamera (const char* connectionType, int32_t portNumber)
C#	int cSharpWrapperApi::loadConfigFromCamera(string conectionType, int portNumber)
	(deprecated) int cSharpWrapperApi::loadConfigFromCamera(sbyte* conectionType, int portNumber)

Input values:

connectionType – string which holds the connection type for connecting to the 3DPIXA. Currently supported connection types are: “RS232” and “CL” (Camera Link).

portNumber – integer which holds the port number for connecting to the camera.

Return values:

0 if no error, <0 otherwise.

Description:

Loads a configuration from camera and sets this configuration as the new configuration in the API.

6.3.8 saveConfigToCamera

C++	int32_t I3DApi::saveConfigToCamera (const char* connectionType, int32_t portNumber)
C#	int cSharpWrapperApi::saveConfigToCamera(string conectionType, int portNumber)
	(deprecated) int cSharpWrapperApi::saveConfigToCamera(sbyte* conectionType, int portNumber)

Input values:

connectionType – string which holds the connection type for connecting to the 3DPIXA. Currently supported connection types are: “RS232” and “CL” (Camera Link).

portNumber – integer which holds the port number for connecting to the camera.

Return values:

0 if no error, <0 otherwise.

Description:

Writes the new configuration to the camera memory. If no new configuration is present, the active configuration will be written.

6.3.9 saveCalibration

C++	int32_t I3DApi::saveCalibration (char* filename)
C#	int cSharpWrapperApi::saveCalibration (String^ filename)

Input values:

filename – string which holds the filename (including path) to write the calibration-file to.

Return values:

0 if no error, <0 otherwise

Description:

Writes the calibration to a given file.

6.3.10 freeConfig

C++	int32_t I3DApi::freeConfig (config3DApi **config)
C#	Not needed

Input values:

C++: config – pointer to a given configuration object.

Return values:

C++: 0 if no error, <0 otherwise.

Description:

C++: Deletes a given configuration object.
C#: freeConfig() is not needed.

6.4 Image Loading

REMARK: Waiting to load next input image can be done with function 6.4.11 actively or via event with WaitForMultipleObjects and function 6.4.4 (Windows only).

On Linux use the acquireImages (6.4.5) function for this purpose.

6.4.1 setSrcImgInfo

C++	int32_t I3DApi::setSrcImgInfo (int32_t camNr, int32_t width, int32_t height, int32_t channels, int32_t bpp, int32_t linePitch, uint64_t sizeInByte)
C#	int cSharpWrapperApi::setSrcImgInfo (int camNr, int width, int height, int channels, int bpp, int linePitch, ulong sizeInByte)

Input values:

camNr – must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the API, so only one image for camNr=0 has to be set.

width – image width in pixels

height – image height in pixels

channels – number of image channels

bpp – bits per pixel. E.g. for an 8-bit RGB image bpp has to be set to 24, for an 8-bit gray image to 8.

pitch – line pitch in bytes

sizeInByte – size of the whole image in bytes

Return values:

0 if no error, <0 otherwise.

Description:

Sets information of input image for further function calls. Has to be set before getDestImgInfo is called. If src image information changes, it will automatically reinitialize the API.

6.4.2 setSrcImgPtr

C++	int32_t I3DApi::setSrcImgPtr (int32_t scamNr, char* ptr)
C#	int cSharpWrapperApi::setSrcImgPtr (int camNr, byte* ptr)

Input values:

camNr - must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the API, so only one image for camNr=0 has to be set.

ptr – pointer to a source image buffer, for the different image formats see chapter 8.1.

Return values:

0 if no error, <0 otherwise.

Description:

Function to set a pointer to an input image buffer for source camNr. Supported image formats are described in chapter 8.1.

6.4.3 setImageMask

C++	int32_t setImageMask(const unsigned char * p)
C#	C++ Only

Input Values:

p – Pointer to the mask image buffer. The input mask image should be single channel grayscale image of type unsigned char.

Return Values:

0 if no error, <0 otherwise.

Description:

This function provides the mask image data to the API. The calculation process will then be performed on pixel area only specific to mask region.

6.4.4 getLoadingEventForSource

C++	HANDLE I3DApi::getLoadingEventForSource (int32_t camNr)
C#	C++ only

Input values:

camNr - must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the API, so only one image for camNr=0 has to be set.

Return values:

HANDLE (MFC) of event for given input channel.

Description:

Gets the handle of the image loading event. The event signals when it is safe to set a pointer to a new input image.

Note: This function is only available for Windows platforms. For Linux platforms please use the acquireImages() function.

6.4.5 acquireImages

C++	int32_t I3DApi::acquireImages(int32_t &curImgNrOut, unsigned char * _imgAPtr, unsigned char * _imgBPtr)
C#	Not available yet

Output values:

curImgNrOut – This variable will hold the internal image number of the input image dataset given to the API for processing.

Input Values:

imgAPtr- Data pointer of the input image 0 or A. For different formats, check chapter 8.1.

imgBPtr- Data pointer of the input image B. In case of using single camera setup, image data will be divided inside API. In that case, data should be given only to imgAPtr and the imgBPtr could be set to NULL.

Return values:

0 if no error, <0 otherwise.

Description: This function combines the functions getLoadingEventForSource, setSrcImgPtr and setSrcImgLoaded as an alternative approach for passing images to the API. It sets the data pointers of the source images to the API and manages the event handle that signals when it is safe to set a pointer of new input image and the event handle that signals that the data pointer is set. This method will block until it can set the image data pointers and it can be cancelled using the cancel method (6.5.5).

6.4.6 blockUntilFetchableResultReady

C++	int32_t blockUntilFetchableResultReady(uint32_t &_currentImageOut, int32_t _timeout)
C#	Not available yet

Output values:

currentImageOut – This function will store the internal image number of the input image dataset given to the API for processing.

Input Values:

_timeout – Timeout in milliseconds, -1 for infinite waiting.

Return value:

If <0 it returns an error code.

Description:

Blocks until next image is calculated or timeout is reached. The call can be canceled by the cancel function (6.5.5).

6.4.7 getResult

C++	int32_t getResult(uint32_t _currentImageNr, void ** dataPtr, outImgType _type)
C#	Not available yet

Input Values:

currentImageNr – Internal image number of the specific dataset for which the output is required.

dataPtr – The pointer to store the calculated output of output type specified in _type.

_type - The type of the output image. It can be:

IMG_OUT_DISP,
IMG_OUT_DISP_8BIT,
IMG_OUT_RGB,
IMG_OUT_BGR,
IMG_OUT_RGBA,
IMG_OUT_BGRA,
IMG_OUT_GRAY,
IMG_OUT_P3D,
IMG_OUT_CONF_8BIT,
IMG_OUT_CONF_FLOAT,
IMG_OUT_BGR2,

IMG_OUT_GRAY2,
IMG_OUT_RGB2,
IMG_OUT_BGR_PLANES,
IMG_OUT_RGB_PLANES,
IMG_OUT_BGR2_PLANES,
IMG_OUT_RGB2_PLANES
Detailed description see chapter 8.2.

Description:

This function calculates and allocates the memory required to store output of specified type. The calculated output data will then be copied to the pointer. This function provides the output of any calculated image dataset when requested with specific internal image number in _currentImageNr.

Important: Please note that the allocated memory has to be freed explicitly with the freeResult function after it was used.

6.4.8 freeResult

C++	int32_t freeResult(uint32_t imageNr)
C#	Not available yet

Input Values:

imageNr – Internal Image number of the specific dataset that should be freed.

Description:

This function frees internal memory of a calculation result that was earlier requested with the getResult function.

6.4.9 blockUntilImageIsLoaded

C++	C# only
C#	bool cSharpWrapperApi::blockUntilImageIsLoaded (int numCams)

Input values:
numCams – 1 for single camera system and 2 for two camera system

Return values:
C#: true if the image of the camera(s) is loaded, otherwise false.

Description:
Does the task of getLoadingEventForSource and then block the process until the image is loaded

6.4.10 setSrcImgLoaded

C++	int32_t I3DApi::setSrcImgLoaded (int32_t camNr)
C#	int cSharpWrapperApi:: setSrcImgLoaded (int camNr)

Input values:

camNr - must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the API, so only one image for camNr=0 has to be set.

Return values:

0 if no error, <0 otherwise.

Description:

The function signals the end of the image loading process to the API. Has to be called after the pointer to the source image is set with the setSrcImgPtr-function (6.4.2) and the processing has started with start-function (6.5.1)

6.4.11 getSrcImgStatus

C++	sopStates I3DApi::getSrcImgStatus (int32_t camNr)
C#	sopStates cSharpWrapperApi:: getSrcImgStatus (int camNr)

Input values:

camNr - must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the API, so only one image for camNr=0 has to be set.

Return value:

status of image processing, can be

SOP_FREE or

SOP_IMG_LOADED

Description:

The function gets the status of the input image processing for source camNr. If it's SOP_FREE, the input image has been processed and I3DApi::setSrcImgPtr can be set to next image buffer. When the function returns SOP_IMG_LOADING the image is still in use by the API.

6.4.12 getSrcImgChannelOrder

C++	channelOrder I3DApi::getSrcImgChannelOrder (int32_t camNr)
C#	cSharpWrapper::channelOrder cSharpWrapperApi:: getSrcImgChannelOrder (int camNr)

Input values:

camNr – must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the API, so only one image for camNr=0 has to be set.

Return values:

<0 if an error occurs.

Otherwise:

CO_RGB, CO_BGR, CO_RGBA, CO_BGRA, CO_GRAY, CO_BGR_PLANES or CO_RGB_PLANES

Description:

Function gets the channel order of the input image. An explanation of the different channel orders can be found at chapter 8.1.

6.4.13 setSrcImgChannelOrder

C++	int32_t I3DApi:: setSrcImgChannelOrder (int32_t camNr, channelOrder co)
C#	int cSharpWrapperApi:: setSrcImgChannelOrder (int camNr, cSharpWrapper::channelOrder co)

Input values:

camNr – must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems, the image is split within the api, so only one image for camNr=0 has to be set.

co – channel order of the input image, can be

CO_RGB,
CO_BGR,
CO_RGBA,
CO_BGRA,
CO_GRAY,
CO_BGR_PLANES,
CO_RGB_PLANES

Return values:

0 if no error, <0 otherwise.

Description:

Function sets the channel order of the input image. An explanation of the different channel orders can be found at chapter 8.1.

6.4.14 getRoi

C++	int32_t I3DApi::getRoi (int32_t &x, int32_t &y, int32_t &width, int32_t &height)
C#	int cSharpWrapperApi:: getRoi (int* x, int* y, int* width, int* height)

Input values:

x – Holds the horizontal starting position of the ROI in the first image, after call.

y – Holds the vertical starting position of the ROI in the first image, after call.

width – Holds the width of the ROI in the first image, after call.

height – Holds the height of the ROI in the first image, after call.

Return values:

0 if no error, <0 otherwise.

Description:

Function gets a previously set ROI. In case of error, all values are set to “-1” on return.

6.4.15 setRoi

C++	int32_t I3DApi::setRoi (int32_t x, int32_t y, int32_t width, int32_t height)
C#	int cSharpWrapperApi:: setRoi (int x, int y, int width, int height)

Input values:

x – The horizontal starting position of the ROI in the first image.

y – The vertical starting position of the ROI in the first image.

width – The width of the ROI in the first image.

height – The height of the ROI in the first image.

Return values:

0 if no error, <0 otherwise.

Possible errors:

```
CS3D_CORE_NOT_INITIALISED
CS3D_SYSTEM_ALREADY_STARTED
CS3D_INVALID_ROI
CS3D_WARN_ROI_ADJUSTED
```

Description:

Function sets a ROI used for the calculation. The ROI has to be set after initialization and before starting the calculation. Setting the ROI modifies the active configuration. There is a minimum requirement on the size of the ROI of 100x100 px.

6.4.16 moveRoi

C++	int32_t I3DApi::moveRoi (int32_t x, int32_t y)
C#	int cSharpWrapperApi:: moveRoi (int x, int y)

Input values:

x – The horizontal starting position of the roi in the first image.

y – The vertical starting position of the roi in the first image.

Return values:

0 if no error, <0 otherwise.

Possible errors:

```
CS3D_CORE_NOT_INITIALISED
CS3D_INVALID_ROI
CS3D_WARN_ROI_ADJUSTED
```

Description:

Moves the ROI to the given position while the calculation is running. This function has a direct impact on the calculation. It should be only called when no images are in the calculation queue.

6.4.17 enableRoiCalculation

C++	int32_t I3DApi::enableRoiCalculation (void)
C#	int cSharpWrapperApi:: enableRoiCalculation (void)

Input values:

None.

Return values:

0 if no error, <0 otherwise.

Description:

Enables the use of a previously defined ROI. Can’t be called while the calculation is running. Therefore the API has to be stopped before the function can be called.

6.4.18 disableRoiCalculation

C++	int32_t I3DApi::disableRoiCalculation (void)
C#	int cSharpWrapperApi:: disableRoiCalculation (void)

Input values:

None.

Return values:

0 if no error, <0 otherwise.

Description:

Disables the use of a previously defined ROI. Can’t be called while the calculation is running. Therefore the API has to be stopped before the function can be called.

6.5 Calculation Process

6.5.1 start

C++	int32_t I3DApi::start (void)
C#	int cSharpWrapperApi:: start (void)

Input values:
None

Return values:

0 if no error, <0 otherwise.

Description:

Starts the calculation of 3D information. Source image pointer and source image information must be set before via setSrcImgPtr (6.4.2).

6.5.2 stopBlocking

C++	int32_t I3DApi::stopBlocking (void)
C#	int cSharpWrapperApi:: stopBlocking (void)

Input values:
None

Return values:
0 if no error, <0 otherwise.

Description:
Stops processing of additional images and blocks until calculation of the current image pair is finished.

6.5.3 getDestImgInfo

C++	int32_t I3DApi::getDestImgInfo (outImgType imgType, int32_t &width, int32_t &height, int32_t &channels, uint64_t &sizeInByte)
C#	Int cSharpWrapperApi::getDestImgInfo (cSharpWrapper::outImgType imgType, int *width, int* height, int* channels, ulong* sizeInByte)

Input values:
imgType – the type of the output image can be:

IMG_OUT_DISP,
IMG_OUT_DISP_8BIT,
IMG_OUT_RGB,
IMG_OUT_BGR,
IMG_OUT_RGBA,
IMG_OUT_BGRA,
IMG_OUT_GRAY,
IMG_OUT_P3D,
IMG_OUT_CONF_8BIT,
IMG_OUT_CONF_FLOAT,
IMG_OUT_BGR2,
IMG_OUT_GRAY2,
IMG_OUT_RGB2,
IMG_OUT_BGR_PLANES,
IMG_OUT_RGB_PLANES,
IMG_OUT_BGR2_PLANES,
IMG_OUT_RGB2_PLANES
Detailed description see chapter 8.2.

width – contains the image width of the output image in pixels/points after call
height – contains the image height of the output image in pixels/points after call
channels – contains the number of channels after the call
sizeInByte – contains the size of the whole output image in bytes after the call

Return values:

0 if no error, <0 otherwise.

Width, height, channels and size hold “-1” if the given outImgType is not available or no source image information is set.

Description:

Gets information of the output image of the type imgType. Source image information must be set via I3DApi::setSrcImgInfo (6.4.1) before using this function.

6.5.4 getNextImgBlocking

Variant 1: getNextImgBlocking without parameter

C++	int32_t I3DApi::getNextImgBlocking (void)
C#	long cSharpWrapperApi:: getNextImgBlocking (void)

Input values:

None

Return value:

If <0 it returns an error code, otherwise the current image number.

Description:

Blocks until next image is calculated. It behaves like variant 2 with a timeout = -1. The call can be canceled by the cancel-function (6.5.5).

Variant 2: getNextImgBlocking with parameter

C++	int32_t I3DApi::getNextImgBlocking (int32_t timeout)
C#	long cSharpWrapperApi:: getNextImgBlocking (int timeout)

Input values:

timeout – Timeout in milliseconds, -1 for infinite waiting.

Return value:

If <0 it returns an error code, otherwise the current image number.

Description:

Blocks until next image is calculated or timeout is reached. The call can be canceled by the cancel function (6.5.5).

6.5.5 cancel

C++	int32_t I3DApi::cancel (void)
C#	int cSharpWrapperApi:: cancel (void)

Input values:

None

Return values:

0 if no error, <0 otherwise.

Description:

Makes the functions getNextImgBlocking (6.5.4) and acquireImages (6.4.5) return at once. If getNextImgBlocking is not running when this function is called, the cancel command is preserved for the next getNextImgBlocking call, and will become invalid after calling the 6.5.2 stopBlocking() function. The calculation of the image is not affected.

6.5.6 getLastImage

6.5.6.1 getLastImage for 16-bit height map images

Variant 1: linePitch is not given.

C++	int32_t I3DApi::getLastImage (int16_t ** imgPtr, outImgType imgType)
C#	int cSharpWrapperApi:: getLastImage (cSharpWrapper::outImgType imgType, ushort** imgPtr)

Input values:

imgPtr – unsigned short pointer to allocated buffer, for image format see chapter 8.2.

imgType – the type of the output image can be:

IMG_OUT_DISP

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function copies the destination image to buffer ptr. The buffer must be allocated by the user. You can get the buffer size and the additional destination image information via the getDestImgInfo-function. The output image types are described in chapter 8.2.

Variant 2: linePitch is given.

C++	int32_t I3DApi::getLastImage (int16_t** imgPtr, int32_t linePitch, outImgType imgType)
C#	int cSharpWrapperApi:: getLastImage (cSharpWrapper::outImgType imgType, int linePitch, ushort** imgPtr)

Input values:

imgPtr – unsigned short pointer to allocated buffer, for image format see chapter 8.2.

linePitch – length of an image line in bytes, including padding bytes.

imgType – the type of the output image can be:

IMG_OUT_DISP

Return values:

Function returns 0 on no error, <0 otherwise

Description:

This function copies the destination image to buffer ptr. The buffer must be allocated by the user. You can get the buffer size and the additional destination image information via getDestImgInfo function. The output image types are described in chapter 8.2.

6.5.6.2 getLastImage for rectified image or 8-bit height map or 8-bit confidence map

Variant 1: linePitch is not given.

C++	int32_t I3DApi::getLastImage (int16_t** imgPtr, outImgType imgType)
C#	int cSharpWrapperApi:: getLastImage (cSharpWrapper::outImgType imgType, byte** imgPtr)

Input values:

imgPtr – unsigned char pointer to allocated buffer, for image format see chapter 8.2.

imgType – the type of the output image can be:

IMG_OUT_RGB,
IMG_OUT_BGR,
IMG_OUT_RGBA,
IMG_OUT_BGRA,
IMG_OUT_DISP_8BIT,
IMG_OUT_GRAY,
IMG_OUT_CONF_8BIT,
IMG_OUT_BGR2,
IMG_OUT_GRAY2,
IMG_OUT_RGB2,
IMG_OUT_BGR_PLANES,
IMG_OUT_RGB_PLANES,
IMG_OUT_BGR2_PLANES,
IMG_OUT_RGB2_PLANES

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function copies the destination image to the buffer ptr. The buffer must be allocated by the user. You can get the buffer size and the additional destination image information the getDestImgInfo function. The output image types are described in chapter 8.2.

Variant 2: linePitch is given.

C++	int32_t I3DApi::getLastImage (int16_t** imgPtr, int32_t linePitch, outImgType imgType)
C#	int cSharpWrapperApi:: getLastImage (cSharpWrapper::outImgType imgType, int linePitch, byte** imgPtr)

Input values:

imgPtr – unsigned char pointer to allocated buffer, for image format see chapter 8.2.

linePitch – length of an image line in bytes, including padding bytes.

imgType – the type of the output image can be:

IMG_OUT_RGB,
IMG_OUT_BGR,

IMG_OUT_RGBA,
IMG_OUT_BGRA,
IMG_OUT_DISP_8BIT,
IMG_OUT_GRAY,
IMG_OUT_CONF_8BIT,
IMG_OUT_BGR2,
IMG_OUT_GRAY2,
IMG_OUT_RGB2,
IMG_OUT_BGR_PLANES,
IMG_OUT_RGB_PLANES,
IMG_OUT_BGR2_PLANES,
IMG_OUT_RGB2_PLANES

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function copies the destination image to the buffer ptr. The buffer must be allocated by the user. You can get the buffer size and the additional destination image information via the getDestImgInfo function. The output image types are described in chapter 8.2.

6.5.6.3 getLastImage for point cloud or float confidence map image

Variant 1: linePitch is not given.

C++	int32_t I3DApi::getLastImage (float** imgPtr, outImgType imgType)
C#	int cSharpWrapperApi:: getLastImage (cSharpWrapper::outImgType imgType, float** imgPtr)

Input values:

imgPtr – float pointer to allocated buffer, for image format see chapter 8.2.

imgType – the type of the output image can be:

IMG_OUT_P3D,
IMG_OUT_CONF_FLOAT

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function copies the destination image to buffer ptr. The buffer must be allocated by the user. You can get the buffer size and the additional destination image information via the getDestImgInfo function. The output image types are described in chapter 8.2.

Variant 2: linePitch is given.

C++	int32_t I3DApi::getLastImage (float** imgPtr, int32_t linePitch, outImgType imgType)
C#	int cSharpWrapperApi:: getLastImage (cSharpWrapper::outImgType imgType, int linePitch, float** imgPtr)

Input values:

imgPtr – float pointer to allocated buffer, for image format see chapter 8.2.

linePitch – length of an image line in bytes, including padding bytes.

imgType – the type of the output image can be:

IMG_OUT_P3D,
IMG_OUT_CONF_FLOAT

Return values:

Function returns 0 on no error, <0 otherwise

Description:

This function copies the destination image to buffer ptr. The buffer must be allocated by the user. You can get the buffer size and the additional destination image information via the function. The output image types are described in chapter 8.2.

6.5.7 freeDestImage

C++	int32_t I3DApi::freeDestImage (void)
C#	int cSharpWrapperApi:: freeDestImage (void)

Input values:

None

Return values:

0 if no error, <0 otherwise.

Description:

This function explicitly frees the internal buffer of the last image. It is also done automatically if I3Dapi::getNextImgBlocking or cSharpWrapperApi::getNextImgBlocking is called the next time.

6.5.8 rawImageCoordsTo3D

Variant 1: Automatically detect the image height

C++	int32_t I3Dapi::rawImageCoordsTo3D(int32_t numPoints, channel chan, float* imgAXPos, float* imgAYPos, float* imgBXPos, float* imgBYPos, float*** resXYZPos)
C#	int cSharpWrapperApi::rawImageCoordsTo3D(int numPoints, cSharpWrapper::channel chan, float[] imgAXPos, float[] imgAYPos, float[] imgBXPos, float[] imgBYPos, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given array.

chan – Channel of the points given.

CHAN_BLUE,
CHAN_GREEN,
CHAN_RED

imgAXPos – Array of column positions of source image A.

imgAYPos – Array of row positions of source image A.

imgBXPos – Array of column positions of source image B.

imgBYPos – Array of row positions of source image B.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, the parameter resXYZPos is not a pointer but a 2 dimensional float array with a size of [numPoints, 3]. The calculated coordinates in resXYZPOs have millimeter as unit.

Return values:

Function returns 0 on no error, <0 otherwise.

When function returns `CS3D_WARN_P3D_NOT_VALID`, one or more pairs of raw coordinates have invalid 3D calculation results, whose results are set to all-zero triplet.

Description:

This function calculates 3D world coordinates in mm for each set of 2D pixel coordinates that are passed to it. With that function you can get 3D coordinates from manual determined 2D coordinates. After a successful execution the resXYZPos array will hold the X, Y and Z coordinates for each given set of coordinates.

Variant 2: Manually set the image height

C++	int32_t I3Dapi::rawImageCoordsTo3D(int32_t numPoints, channel chan, int32_t imageHeight, float* imgAXPos, float* imgAYPos, float* imgBXPos, float* imgBYPos, float*** resXYZPos)
C#	int cSharpWrapperApi::rawImageCoordsTo3D(int numPoints, cSharpWrapper::channel chan, int imageHeight, float[] imgAXPos, float[] imgAYPos, float[] imgBXPos, float[] imgBYPos, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given array.

chan – Channel of the points given.

CHAN_BLUE,
CHAN_GREEN,
CHAN_RED

imageHeight – Image height to use for the calculation of the y coordinates

imgAXPos – Array of column positions of source image A.

imgAYPos – Array of row positions of source image A.

imgBXPos – Array of column positions of source image B.

imgBYPos – Array of row positions of source image B.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, the parameter resXYZPos is not a pointer but a 2 dimensional float array with a size of [numPoints, 3]. The calculated coordinates in resXYZPOs have millimeter as unit.

Return values:

Function returns 0 on no error, <0 otherwise

When function returns `CS3D_WARN_P3D_NOT_VALID`, one or more pairs of raw coordinates have invalid 3D calculation results, whose results are set to all-zero triplet.

Description:

This function behaves like variant 1, but with a specific image height given that is used to calculate the y coordinates.

6.5.9 rectImageCoordsTo3D

Variant 1: Automatically detect the image height

C++	int32_t I3Dapi::rectImageCoordsTo3D(int32_t numPoints, float* imgAXPos, float* imgAYPos, float* imgBXPos, float* imgBYPos, float*** resXYZPos)
C#	int cSharpWrapperApi::rectImageCoordsTo3D (int numPoints, float[] imgAXPos, float[] imgAYPos, float[] imgBXPos, float[] imgBYPos, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given array.

imgAXPos – Array of column positions of rectified image A.

imgAYPos – Array of row positions of rectified image A.

imgBXPos – Array of column positions of rectified image B.

imgBYPos – Array of row positions of rectified image B.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, the parameter resXYZPos is not a pointer but a 2 dimensional float array with a size of [numPoints, 3]. The calculated coordinates in resXYZPOs have millimeter as unit.

Return values:

Function returns 0 on no error, <0 otherwise.

When function returns `CS3D_WARN_P3D_NOT_VALID`, one or more pairs of raw coordinates have invalid 3D calculation results, whose results are set to all-zero triplet.

Description:

This function calculates 3D world coordinates in mm for each set of 2D pixel coordinates that are passed to it. With that function you can get 3D coordinates from manual determined 2D coordinates. After a successful execution the resXYZPos array will hold the X, Y and Z coordinates for each given set of coordinates.

Variant 2: Manually set the image height

C++	int32_t I3Dapi::rectImageCoordsTo3D(int32_t numPoints, int32_t imageHeight, float* imgAXPos, float* imgAYPos, float* imgBXPos, float* imgBYPos, float*** resXYZPos)
C#	int cSharpWrapperApi::rectImageCoordsTo3D(int numPoints, int imageHeight, float[] imgAXPos, float[] imgAYPos, float[] imgBXPos, float[] imgBYPos, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given array.

imageHeight – Image height to use for the calculation of the y coordinates

imgAXPos – Array of column positions of rectified image A.

imgAYPos – Array of row positions of rectified image A.

imgBXPos – Array of column positions of rectified image B.

imgBYPos – Array of row positions of rectified image B.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, the parameter resXYZPos is not a pointer but a 2 dimensional float array with a size of [numPoints, 3]. The calculated coordinates in resXYZPOs have millimeter as unit.

Return values:

Function returns 0 on no error, <0 otherwise

When function returns `CS3D_WARN_P3D_NOT_VALID`, one or more pairs of raw coordinates have invalid 3D calculation results, whose results are set to all-zero triplet.

Description:

This function behaves like variant 1, but with a specific image height given that is used to calculate the y coordinates.

6.5.10 rawImageCoordsToRectCoords

C++	int32_t I3Dapi::rawImageCoordsToRectCoords(float ** rectXPosOut, float ** rectYPosOut, int32_t numPoints, channel chan, float* imgXPos, float* imgYPos, int32_t cameraNr)
C#	int cSharpWrapperApi:: rawImageCoordsToRectCoords (ref float[] rectXPosOut, ref float[] rectYPosOut, int numPoints, channel chan, int numPoints, float[] rawXPos, float[] rawYPos, int cameraNr)

Input values:

rectXPosOut – Pointer to an allocated float array, that must have [numPoints] dimensions.

rectYPosOut – Pointer to an allocated float array, that must have [numPoints] dimensions.

numPoints – Number of points within the given array.

chan – Channel of the points given.

CHAN_BLUE,
CHAN_GREEN,
CHAN_RED

rawXPos – Array of column positions of the raw image.

rawYPos – Array of row positions of raw image.

cameraNr - Must be 0 for the image of camera A or 1 for the image of camera B. In single camera systems only camNr=0 has to be set.

Return values:

Function returns 0 on no error, <0 otherwise

Description:

This function calculates 2D rectified image coordinates in pixel for each set of raw image 2D pixel coordinates that are passed to it. After a successful execution the rectXPosOut /rectYPosOut array will hold the X and Y coordinates for each given set of coordinates. The central view correction will not apply to the results.

6.5.11 grayTo3D

6.5.11.1 grayTo3D for 16-Bit gray values

Variant 1: Automatically detect the image height

C++	int32_t I3Dapi::grayTo3D(int32_t numPoints, int32_t * imgXPos, int32_t * imgYPos, uint16_t *SgrayVals, float *** resXYZPos)
C#	int cSharpWrapperApi::grayTo3D(int numPoints, int[] imgXPos, int[] imgYPos, ushort[] grayVals, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given arrays.

imgXPos – Array of column positions of height image.

imgYPos – Array of row positions of height image.

grayVals – Array of gray values.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, resXYZPos is not a pointer but a 2 dimensional float array, which has the size of [numPoints, 3]

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function calculates 3D world coordinates for each set of combinations of X,Y coordinates and gray values from a height image.

Variant 2: Manually set the image height

C++	int32_t I3Dapi::grayTo3D(int32_t numPoints, int32_t imageHeight, int32_t * imgXPos, int32_t * imgYPos, uint16_t* grayVals, float*** resXYZPos)
C#	int cSharpWrapperApi::grayTo3D(int numPoints, int imageHeight, int[] imgXPos, int[] imgYPos, ushort[] grayVals, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given arrays.

imageHeight – Image height to use for the calculation of the y coordinates

imgXPos – Array of column positions of height image.

imgYPos – Array of row positions of height image.

grayVals – Array of gray values.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, resXYZPos is not a pointer but a 2 dimensional float array, which has a size of [numPoints, 3]

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function calculates 3D coordinates for each set of combinations of X,Y coordinates and gray values from a height image.

6.5.11.2 grayTo3D for 8-Bit gray values

Variant 1: Automatically detect the image height

C++	int32_t I3Dapi::grayTo3D(int32_t numPoints, int32_t * imgXPos, int32_t * imgYPos, unsigned char* grayVals, float*** resXYZPos)
C#	int cSharpWrapperApi::grayTo3D(int numPoints, int[] imgXPos, int[] imgYPos, byte[] grayVals, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given arrays.

imgXPos – Array of column positions of height image.

imgYPos – Array of row positions of height image.

grayVals – Array of gray values.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, resXYZPos is not a pointer but a 2 dimensional float array, which has a size of [numPoints, 3]

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function calculates 3D coordinates for each set of combinations of X,Y coordinates and gray values from a height image.

Variant 2: Manually set the image height

C++	int32_t I3Dapi::grayTo3D(int32_t numPoints, int32_t imageHeight, int32_t * imgXPos, int32_t * imgYPos, unsigned char* grayVals, float*** resXYZPos)
C#	int cSharpWrapperApi::grayTo3D(int numPoints, int imageHeight, int[] imgXPos, int[] imgYPos, byte[] grayVals, ref float[,] resXYZPos)

Input values:

numPoints – Number of points within the given arrays.

imageHeight – Image height to use for the calculation of the y coordinates.

imgXPos – Array of column positions of height image.

imgYPos – Array of row positions of height image.

grayVals – Array of gray values.

resXYZPos – Pointer to an allocated 2 dimensional float array, that must have [numPoints][3] dimensions. In C#, resXYZPos is not a pointer but a 2 dimensional float array, which has a size of [numPoints, 3]

Return values:

Function returns 0 on no error, <0 otherwise

Description:

This function calculates 3D coordinates for each set of combinations of X and Y coordinates and gray values from a height image.

6.6 Miscellaneous

This chapter describes functions for miscellaneous use.

6.6.1 getVersion

C++	void I3Dapi::getVersion (int32_t &major, int32_t &minor, int32_t &build)
C#	void cSharpWrapperApi:: getVersion (int* major, int* minor, int* build)

Input values:
major – contains major version number after the call.
Minor – contains minor version number after the call.
Build – contains build number after the call.

Return values:
None.

Description:
Return the version of the API.

6.6.2 isStarted

C++	bool I3Dapi::isStarted (void)
C#	bool cSharpWrapperApi:: isStarted(void)

Input values:
None.

Return values:
True if the API is started, otherwise false

Description:
Returns the status of the API

6.6.3 isInitialized

C++	bool I3Dapi::isInitialized (void)
C#	bool cSharpWrapperApi:: isInitialized(void)

Input values:

None.

Return values:

True if the API is initialized, otherwise false

Description:

Returns the status of the API

6.6.4 grayToMm

Variant 1: grayToMm with for 16-bit gray values

C++	int32_t I3Dapi::grayToMm (float &distInMm, unsigned short grayValue)
C#	int cSharpWrapperApi:: grayToMm (float* distInMm, ushort grayValue)

Input values:

distInMm – float to store the distance value in mm, which is the distance between object surface and camera.

grayValue – the 16-bit intensity value in height map to be converted into distance value in mm.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function converts the intensity values to distance values in mm, based on the configuration/calibration that is currently loaded. Especially, the upper / lower height range limit of the configuration and the calibration data must match with the values used for the calculation of the height map image.

Variant 2: grayToMm for 8-bit gray values

C++	int32_t I3Dapi::grayToMm (float &distInMm, unsigned char grayValue)
C#	int cSharpWrapperApi:: grayToMm (float* distInMm, byte grayValue)

Input values:

dispInMm – float that stores the distance value in mm, which is the distance between object surface and camera.

grayValue – the 8-bit intensity value in height map to be converted into distance value in mm.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function behaves like grayToMm variant 1, except that the gray value is in 8-bit.

6.6.5 dispToMm

Variant 1: dispToMm with integer relative disparity values

C++	int32_t I3Dapi::dispToMm (float &distInMm, int32_t dispValue, bool absoluteDistance=true)
C#	int cSharpWrapperApi:: dispToMm (float* distInMm, int dispValue, bool absoluteDistance)

Input values:

distInMm – float to store the distance value in mm, which is the distance between object surface and camera or object surface and FWD.

dispValue – the relative disparity value, which refers to the difference in pixels of same physical locations between two stereo images. The relative disparity value of locations at working distance is around 0. For locations further or closer to the camera than the working distance, the relative disparity value is negative or positive.

absoluteDistance – optional, default is true, means distance to camera is returned. False means the distance to a plane in FWD is returned.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function converts a relative disparity value in pixels to a distance in mm, based on the configuration/calibration that is currently loaded. This function can be used for instance to convert the lower height range limit in pixels into a distance to the camera or into a distance to the plane in FWD.

Variant 2: dispToMm with float relative disparity values

C++	int32_t I3Dapi::dispToMm (float &distInMm, float dispValue, bool absoluteDistance=true)
C#	int cSharpWrapperApi:: dispToMm (float* distInMm, float dispValue, bool absoluteDistance)

Input values:

distInMm – float to store the distance value in mm, which is the distance between object surface and camera.

dispValue – the relative disparity value, which refers to the difference in pixels of same physical locations between two stereo images. The relative disparity value of locations at working distance is around 0. For locations further or closer to the camera than the working distance, the relative disparity value is negative or positive.

absoluteDistance – optional, default is true, means distance to camera is returned. False means the distance to a plane in FWD is returned.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function behaves like dispToMm variant 1, except that the relative disparity value is float.

6.6.6 mmToDisp

Variant 1: mmToDisp with integer relative disparity values

C++	int32_t I3Dapi::mmToDisp (int32_t &dispValue, float distInMm, bool absoluteDistance=true)
C#	int cSharpWrapperApi:: mmToDisp (int* dispValue, float distInMm, bool absoluteDistance)

Input values:

dispValue –int to store the relative disparity value.

distInMm – distance value to be converted into relative disparity value.

absoluteDistance – optional, default is true, means distance to camera is given. False means the distance to a plane in FWD is given.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function converts a distance value in mm into a relative disparity value in pixels, based on the configuration/calibration that is currently loaded. This function can be used for instance to convert a distance to the camera or a distance to the plane in WD/FWD into a value that can be used as the upper height range limit in pixels.

Variant 2: mmToDisp with float relative disparity values

C++	int32_t I3Dapi::mmToDisp (float &dispValue, float distInMm, bool absoluteDistance=true)
C#	int cSharpWrapperApi:: mmToDisp (float* dispValue, float distInMm, bool absoluteDistance)

Input values:

dispValue –float to store the relative disparity value.

distInMm – distance value to be converted into relative disparity value.

absoluteDistance – optional, default is true, means distance to camera is given. False means the distance to a plane in FWD is given.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function behaves like mmToDisp variant 1, except that the relative disparity value is float.

6.6.7 setSrcImgIndex

C++	int32_t I3Dapi::setSrcImgIndex (uint32_t SimgNr)
C#	int cSharpWrapperApi:: setSrcImgIndex (uint imgNr)

Input values:

imgNr – Start image number to use.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function sets the internal image number that is also returned by the getNextImgBlocking function to the given value. If not set, the image counter starts with zero. This function has to be called AFTER the API is initialized. Initialize or reinitialize functions will reset the image counter to zero.

6.6.8 getApiMemoryInformation

C++	int32_t I3Dapi::getApiMemoryInformation (config3Dapi* cfg, int32_t imgWidth, int32_t imgHeight, int32_t &cpuRamNeeded, int32_t &gpuRamNeeded)
C#	int cSharpWrapperApi::getApiMemoryInformation (int imgWidth, int imgHeight, int* cpuRamNeeded, int* gpuRamNeeded)
	int cSharpWrapperApi::getApiMemoryInformationFromConfig (cSharpWrapper::CS3Dconfig cfg, int imgWidth, int imgHeight, int* cpuRamNeeded, int* gpuRamNeeded)

Input values:

cfg – configuration to use for the estimation of memory usage. If this parameter is not given in C#, the current configuration will be used for estimation.

imgWidth – source image width.

imgHeight – source image height.

cpuRamNeeded – integer to store needed CPU memory in MB.

gpuRamNeeded – integer to store needed GPU memory in MB.

Return values:

Function returns 0 on no error, <0 otherwise.

Description:

This function provides estimated CPU and GPU RAM usage for a given configuration and image height and width.

For the C#-version, if a configuration is not given, then the current active configuration file is used for the estimation. Otherwise the estimation will be based on the given configuration.

6.6.9 verifyCalibration

Variant 1: using pointer of source image buffer as a parameter

C++	int32_t I3Dapi::verifyCalibration (unsigned char* ptrA, unsigned char* ptrB = NULL)
C#	int cSharpWrapperApi::verifyCalibration(byte* imPtrA, byte* imPtrB)

Input values:

ptrA – pointer to a source image buffer from camera A.

ptrB – pointer to a source image buffer from camera B. By default, this pointer is set to NULL in case of single camera systems.

Return values:

0 if no error, <0 otherwise.

Possible errors:

```
CS3D_SUCESS
CS3D_CANT_START_SYSTEM
CS3D_FUNCTION_NA
CS3D_WARN_CALIB_NOT_VALID_ANYMORE
CS3D_CALIB_VERIFICATION_FAILED
```

Description:

This function verifies if the current calibration is still valid for the current status of the camera. Raw image data will be passed as parameter, so please make sure to call setSrcImgInfo() (6.4.1) previously before using verifyCalibration().

Variant 2: using source image filename as parameter

C++	int32_t I3Dapi::verifyCalibrationWithFileName (const char* cameraAFilename, const char* cameraBFilename = NULL)
C#	int cSharpWrapperApi::verifyCalibrationWithImageFileName (string cameraAFilename, string cameraBFilename)

Input values:

cameraAFilename – filename (including path) from camera A, which will be used for verification.

cameraBFilename – filename (including path) from camera B, which will be used for verification. By default, this pointer is set to NULL in case of single camera systems.

Return values:

0 if no error, <0 otherwise.

Description:

Just like verifyCalibration() in 6.6.9, this function verifies if the current calibration is still valid for the current status of the camera.

6.6.10 adjustCalibration

Variant 1: overwite the old calibration file

C++	int32_t I3Dapi::adjustCalibration (unsigned char* ptrA, unsigned char* ptrB = NULL)
C#	int cSharpWrapperApi::adjustCalibration(byte* imPtrA, byte* imPtrB)

Input values:

ptrA – pointer to a source image buffer from camera A.

ptrB – pointer to a source image buffer from camera B. By default, this pointer is set to NULL in case of single camera systems.

Return values:

0 if no error, <0 otherwise.

Possible errors:

```
CS3D_SUCESS
CS3D_SYSTEM_ALREADY_STARTED
CS3D_CANT_START_SYSTEM
CS3D_FUNCTION_NA
CS3D_CALIB_VERIFICATION_FAILED
```

Description:

This function does not only verify the calibration, but also adjust the calibration to improve the calculation precision afterwards. The old calibration file will be overwritten after execution. For the same reason like verifyCalibration 6.6.9, setSrcImgInfo() has to be called previously.

Variant 2: create a new calibration file instead of overwriting the old one

C++	int32_t I3Dapi::adjustCalibration (const char* targetFilename, unsigned char* ptrA, unsigned char* ptrB = NULL)
C#	int cSharpWrapperApi::adjustCalibration(string targetFilename, byte* imPtrA, byte* imPtrB)

Input values:

targetFilename – filename (incl. path) of the generated new calibration file.

ptrA – pointer to a source image buffer from camera A.

ptrB – pointer to a source image buffer from camera B. By default, this pointer is set to NULL in case of single camera systems.

Return values:

0 if no error, <0 otherwise.

Description:

Just like variant 1 of adjustCalibration(), variant 2 adjusts the calibration and saves the generated file to a new file instead of overwriting the old one.

6.6.11 errorToString

C++	int32_t I3Dapi::errorToString (char* errStrBuff, int32_t buffLen, int32_t &errStrLen, int32_t errCode, const char* language)
C#	int cSharpWrapperApi:: errorToString(ref string errStrBuff, int buffLen, ref int errStrLen, int errCode, string language)

Input values:

errStrBuff – in C++, it is a pointer to char array buffer which holds the returned error string. In C#, it is directly the returned error message.

buffLen – size of the char array buffer.

errStrLen – actual length of the returned error string.

errCode – input error code whose corresponding error string is to be queried.

Language – language of the error string. Currently English “EN” and Chinese “ZH” are supported.

Return values:

0 if no error, <0 otherwise.

Description:

This function translates error codes into error message strings in different languages. Currently English “EN” and Chinese “ZH” are supported. When language is set to “ZH”, the returned string is utf8 encoded.

Helper functions for string conversion are available in C++ sample code.

6.6.12 getLicenseInformation

C++	int32_t I3Dapi::getLicenseInformation (CS3DLicenseInfo * licInfo)
C#	int cSharpWrapperApi::getLicenseInformation (CS3DLicenseInfo licInfo)

Input values:

licInfo – Pass an CS3DLicenseInfo object that will be filled with information after the function returns.

Return values:

0 if no error, <0 otherwise.

```
licInfo.validUntil          // string that holds the date of expire, zero terminated
licInfo.multiLicense        // Does the dongle contains multiple licenses or a single
                            license
licInfo.numLicense          // number of licenses
licInfo.serialNumber        // string that holds the serial number, zero terminated
```

Data type of attributes in C++ and C#

	licInfo.validUntil	licInfo.multiLicense	licInfo.numLicense	licInfo.serialNumber
C++	char*	bool	int	char*
C#	string	bool	int	string

Description:

This function gets the licenses information from the connected dongle. If there is more than one dongle connected, the license information from the first is displayed.

6.6.13 staticCalibration

C++	int32_t I3Dapi::staticCalibration (CS3DStaticCalibrationResult * calibRes, unsigned char* ptrA, unsigned char* ptrB = NULL, const char * tDef)

Input values:

calibRes – Pass an CS3DStaticCalibrationResult object that will be filled with information after the function returns.

ptrA – pointer to a source image buffer from camera A.

ptrB – pointer to a source image buffer from camera B. Set to NULL in case of single camera systems.

tDef – filename that holds the definition of the target

Return values:

0 if no error, <0 otherwise.

```
calibRes.tx                 // translation along sensor axis in [mm]
calibRes.ty                 // translation along transport axis in [mm]
calibRes.tz                 // translation along distance axis in [mm]
calibRes.ry                 // beta / rotation around transport axis in [rad]
calibRes.rz                 // gamma / rotation around distance axis in [rad]
calibRes.targetID           // string that holds the ID and serial number of the target
calibRes.rXX                // 4x4 Transformation matrix that transforms each 3DPoint
                            in the coordinate system of the target.
```

$$R = \begin{pmatrix} r_{00} & r_{01} & r_{02} & r_{03} \\ r_{10} & r_{11} & r_{12} & r_{13} \\ r_{20} & r_{21} & r_{22} & r_{23} \\ r_{30} & r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Description:

Currently in BETA state. This function does a static calibration to enable the user to adjust the position of the point cloud. A specific target is needed for this functionality. Be sure to set the image information with setSrcImgInfo() (6.4.1) before using the staticCalibration() function.

7 HALCON – Extension

There are HALCON extensions available for HALCON 12, 13 and 18.11 on Windows platforms. HALCON should be installed before the CS-3D software is installed. To switch between the versions either use the CS-3D software installer or change the path to the CS-3D HALCON Extension as explained in the HALCON documentation.

7.1 HALCON – Operator List

7.1.1 Control

7.1.1.1 create_cs_3d_handle

```
create_cs_3d_handle(‘config_file’, configFile, CS3Dhandle)
```

Input values:

configFile – String that contains path and name of the configuration file.

Return values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Description:

Does the initialization of the API with the configuration from the given configuration file. You can create multiple instances, but for each instance you need one license.

7.1.1.2 clear_cs_3d_handle (CS3Dhandle)

```
clear_cs_3d_handle(CS3Dhandle)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:

None.

Description:

Unloads a given instance and frees its allocated memory.

7.1.2 Configuration

7.1.2.1 cs_3d_load_config_from_camera

```
cs_3d_load_config_from_camera(CS3Dhandle, connectionType, port)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

ConnectionType – String to set the type of connection ‘RS232’ for serial connection via a (virtual) com port. ‘CL’ for a connection via the framegrabber.

Port – Port number used for communication.

Return values:

None.

Description:

This function will load the configuration from a connected 3DPIXA.

7.1.2.2 cs_3d_save_config_to_camera

```
cs_3d_save_config_to_camera(CS3Dhandle, connectionType, port)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

ConnectionType – String to set the type of connection ‘RS232’ for serial connection via a (virtual) com port. ‘CL’ for a connection via the framegrabber.

Port – Port number used for communication.

Return values:

None.

Description:

This function will save the current configuration to a connected 3DPIXA.

7.1.2.3 cs_3d_load_config_from_file

```
cs_3d_load_config_from_file(CS3Dhandle, filename)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Filename – Filename to load the config from.

Return values:

None.

Description:

This function will load the configuration from a given file.

7.1.2.4 cs_3d_save_config_to_file

```
cs_3d_save_config_to_file(CS3Dhandle, filename)
```

Input values:
CS3Dhandle – Holds the handle for an instance of the CS-3D-API.
Filename – File to save the config to.

Return values:
None.

Description:
This function will save the current configuration to a given file.

7.1.2.5 cs_3d_set_param

```
cs_3d_set_param(CS3Dhandle, [ parameterName, … ], [ parameterValue, … ])
```

Input values:
CS3Dhandle – Holds the handle for an instance of the CS-3D-API.
parameterName – Can hold one or a list of the parmeters mention below.
parameterValue – Holds one value for each of the given parameters.

Return values:
None.

Parameter	Type	Example	Corresponding config entry
config_file	String	‘C:/3D/config.ini’	
height_range_start_in_px	Integer	-10	dStart
height_range_end_in_px	Integer	10	dEnd
min_correlation_score	Double	0.5	minBadKkf
max_lr_pixels	Double	10	Consistent
min_intensity	Integer	1	darkR
max_intensity	Integer	254	brightR
min_contrast	Double	0.1	minStdA
window_type	String	‘15x15’	Chapter 10.2.3.1 includes a list of available size
timeout	Integer	20000	Timeout for calculation in [ms]

Description:

Sets one or more parameters to the given values. After that, an automatic reinitialisation is done, so if you like to set more than one parameter it is recommended to do that in a single call of this operator to avoid a reinitialization of the API every time you change a parameter.

If you pass a “config_file” parameter, other changes to the configuration will be ignored to avoid that an accidental overwrite of the configuration is performed.

7.1.2.6 cs_3d_get_param

```
cs_3d_get_param(CS3Dhandle, [ parameterName, … ], resultTuple)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

parameterName – Can hold one or a list of the parmeters mention in chapter 7.1.2.1.

Return values:

resultTuple – Holds one value for each parameter name passed as input.

Description:

This function gets one or more parameters in the configuration of the API instance.

7.1.2.7 cs_3d_roi_set

```
cs_3d_roi_set(CS3Dhandle, y1, x1, y2, x2)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Y1 – Y coordinate of the upper left corner of the ROI

x1 – X coordinate of the upper left corner of the ROI

y2 – Y coordinate of the lower right corner of the ROI

x2 – X coordinate of the lower right corner of the ROI

Return values:

None.

Description:

This function sets a ROI for the calculation of the 3D data. Before the ROI is used it has to be activated using the cs_3d_roi_enable function.

7.1.2.8 cs_3d_roi_get

```
cs_3d_roi_get(CS3Dhandle, y1, x1, y2, x2)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:
y1 – Y coordinate of the upper left corner of the ROI
x1 – X coordinate of the upper left corner of the ROI
y2 – Y coordinate of the lower right corner of the ROI
x2 – X coordinate of the lower right corner of the ROI

Description:
This function returns the parameter for a set ROI.

7.1.2.9 cs_3d_roi_enable

```
cs_3d_roi_enable(CS3Dhandle)
```

Input values:
CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:
None.

Description:
This function enables the usage of a set ROI for the calculation of the 3D data.

7.1.2.10 cs_3d_roi_disable

```
cs_3d_roi_disable(CS3Dhandle)
```

Input values:
CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:
None.

Description:
This function disables the usage of a set ROI for the calculation of the 3D data

7.1.3 Calculation

7.1.3.1 cs_3d_acquire_image

```
cs_3d_acquire_image(Im, CS3Dhandle)
```

Input values:
Im – A single image in case of a compact camera or a tuple of images in case of a dual camera.
CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:

None.

Description:

This function loads the source image(s) into the API. It must be executed before cs_3d_stereo 7.1.3.2. After a parameter is changed, this operator can take longer time because the API is started before the new image is loaded into the API.

7.1.3.2 cs_3d_stereo

```
cs_3d_stereo(Heightmap, Rect, P3D, CS3Dhandle, ObjectModel3D)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:

Heightmap – 16 bit gray scale image that contains the scaled height information. 8.2.2

Rect – 8 bit gray/color rectified image. Number of channels depends on the source input image. 8.2.1

P3D – 3 channel float image that contains the X, Y and Z coordinates of a 3D point cloud. 8.2.3

ObjectModel3D – Contains a 3D Model that can be processed by the sample_object_model_3d HALCON operator.

Description:

This function blocks up to TIMEOUT milliseconds until next results are ready.

7.1.3.3 cs_3d_get_result

```
cs_3d_get_result(ResultImage, CS3Dhandle, ResultType)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

ResultType – Choose one of the types from the list below

ResultType	Data Type	Description
rectified	Image (1or 3x8bit)	Same like the rectified from cs_3d_stereo
rectified_B	Image (1or 3x8bit)	Rectified image from second camera
confidence_map	Image (32bit)	Confidence map according to chapter 8.2.4
height_map	Image (16bit)	Same like the height map from cs_3d_stereo
P3D_array	Image (3x32bit)	Same like the P3D from cs_3d_stereo

Return values:

ResultImage – Holds one or more results.

Description:

This function provides additional results from the 3D calculation. It must be called after cs_3d_stereo (7.1.3.2) and before the next cs_3d_acquire (7.1.3.1).

7.1.3.4 cs _3d_cancel

```
cs_3d_cancel(CS3Dhandle)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:

None.

Description:

Cancels a running calculation. A blocked cs_3d_stereo operator will return immediately.

7.1.3.5 cs_3d_rawimagecoord_to_3d

```
cs_3d_rawimagecoord_to_3d(CS3Dhandle, color_channel, image_height, ay, ax, by, bx, X, Y, Z)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Color_channel – Color channel of the coordinates, can be ‘red’, ‘green’ or ‘blue’. This is only relevant if you use a version 2 calibration. If you use a version 1 calibration just use channel ‘green’.

Image_height – Sets the image height to a specific height, different from the one set in the API. If you want to use the height set in the API use ‘-1’

Ay – One or more row coordinates from image A. Unit is px.

Ax – One or more column coordinates from image A. Unit is px.

By – One or more row coordinates from image B. Unit is px.

Bx – One or more column coordinates from image B. Unit is px.

Return values:

X – The resulting X coordinates. Unit is mm.

Y – The resulting Y coordinates. Unit is mm.

Z – The resulting Z coordinates. Unit is mm.

Description:

This function will generate 3D points from pairs of row/col coordinates of the source images.

7.1.3.6 cs_3d_rectimagecoord_to_3d

```
cs_3d_rectimagecoord_to_3d(CS3Dhandle, image_height, ay, ax, by, bx, X, Y, Z)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Image_height – Sets the image height to a specific height, different from the one set in the API. If you want to use the height set in the API use '-1'

Ay – One or more row coordinates from image A. Unit is px.

Ax – One or more column coordinates from image A. Unit is px.

By – One or more row coordinates from image B. Unit is px.

Bx – One or more column coordinates from image B. Unit is px.

Return values:

X – The resulting X coordinates. Unit is mm.

Y – The resulting Y coordinates. Unit is mm.

Z – The resulting Z coordinates. Unit is mm.

Description:

This function will generate 3D points from pairs of row/col coordinates of the rectified images.

7.1.3.7 cs_3d_rawimagecoord_to_rectcoord

```
cs_3d_rawimagecoord_to_rectcoord(CS3Dhandle, color_channel, rawY, rawX, cameraNr, rectY, rectX)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Color_channel – Color channel of the coordinates, can be 'red', 'green' or 'blue'. This is only relevant if you use a version 2 calibration. If you use a version 1 calibration just use channel 'green'.

rawY – One or more row coordinates from raw image. Unit is px.

rawX – One or more column coordinates from raw image. Unit is px.

cameraNr – Camera number, 0 for master, 1 for slave camera. For compact cameras always 0

Return values:

rectY – The resulting row coordinates. Unit is px.

rectX – The resulting column coordinates. Unit is px.

Description:

This function converts a pair of image coordinates of the raw/source image to a pair of coordinates of the rectified image.

7.1.3.8 cs _3d_gray_to_mm

```
cs_3d_gray_to_mm(CS3Dhandle, GrayValueType, [ GrayValue, … ], DistancesInMm)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

GrayValueType – ‘16bit’ or ‘8bit’, set according to bpp of the height map image the gray value is from.

GrayValue – One or more gray values

Return values:

DistanceInMm – Returns a tuple of distances.

Description:

This function returns for a given gray value the distance to the camera in mm.

7.1.3.9 cs _3d_gray_to_3d

```
cs_3d_gray_to_3d(CS3Dhandle, imageheight, inY, inX, grayType, grayValue, X, Y, Z)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

ImageHeight – Sets the image height to a specific height, different from the one set in the API. If you want to use the height set in the API use ‘-1’

inY – One or more row coordinates from height image. Unit is px.

inX – One or more column coordinates from height image. Unit is px.

GrayType – ‘16bit’ or ‘8bit’, set according to bpp of the height map image the gray value is from.

GrayValue – One or more gray values

Return values:

X – The resulting X coordinates. Unit is mm.

Y – The resulting Y coordinates. Unit is mm.

Z – The resulting Z coordinates. Unit is mm.

Description:

This function returns for given gray values and their positions in the height image the distance to the camera in mm.

7.1.3.10 cs _3d_px_to_mm

```
cs_3d_px_to_mm(CS3Dhandle, DistanceType, PxValue, DistanceInMm)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

DistanceType – ‘distance_to_zero_plane’ means convert the [px] to [mm] relative to the zero plane.
‘distance_FWD’ means convert the [px] to [mm] relative to the FWD position at the camera.

PxValue – Shift in pixels [px]

Return values:

DistanceInMm – Returns a tuple of distances in [mm].

Description:

This function returns for given shifts in [px] the distance of DistanceType. The shift in [px] is “0” at WD/zero-plane, negative below the zero plane and positive above the zero-plane. For further information see chapter 8.2.2, 8.2.3. This function can e.g. be used to convert the start/end of the search range from [px] to [mm].

7.1.3.11 cs _3d_mm_to_px

```
cs_3d_mm_to_px(CS3Dhandle, DistanceType, DistanceInMm, PxValue)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

DistanceType – ‘distance_to_zero_plane’ means convert the [mm] relative to the zero-plane to [px] to.
‘distance_FWD’ means convert the [mm] relative to the FWD position at the camera to [px].

DistanceInMm – Tuple of distances in [mm].

Return values:

PxValue – Shifts in pixels [px]

Description:

This function returns for given shifts in [px] the distance of DistanceType. The shift in [px] is “0” at WD/zero-plane, negative below the zero plane and positive above the zero-plane. For further information see chapter 8.2.2, 8.2.3. This function can e.g. be used to convert the start/end of the search range from [mm] to [px].

7.1.4 Miscellaneous

7.1.4.1 cs _3d_get_version

```
cs_3d_get_version(CS3Dhandle, MajorVersion, MinorVersion, BuildVersion)
```

Input values:

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:

MajorVersion – Major software version.

MinorVersion – Minor software version.

BuildVersion – Software build.

Description:

This function returns information about the CS-3D software version.

7.1.4.2 cs _3d_adjust_calibration

```
cs_3d_adjust_calibration(Im, CS3Dhandle)
```

Input values:

Im – A single image in case of a compact camera or a tuple of images in case of a dual camera.

CS3Dhandle – Holds the handle for an instance of the CS-3D-API.

Return values:

None

Description:

This function adjusts the calibration based on raw image(s) of an object. Overwrites the current calibration.

7.2 .NET/C++ Version of HALCON extension

There is also a C# and a C++ version of the CS-3D HALCON extension available.

The .NET wrapper is available at {appDir}\extensions\HALCON\{HALCON version}\CS3D\bin\dotnet

The C++ version is available at {appDir}\extensions\HALCON\{HALCON version}\CS3D\bin\x64-win64

REMARK: Do NOT copy or move *.dll files to any other location. According to the HALCON documentation this will cause in a non-function HALCON extension. If you need to put the program that depends on that dlls outside this directory, Please add the {appDir}\extensions\HALCON\{HALCON version}\CS3D\bin\x64-win64 to the system PATH variable.

8 Image Format

8.1 *Source Image Format*

The image starts with the upper left pixel. We assume a number of width * height pixels with bpp bits per pixel.

8.1.1 Single Channel Image

8-bit single channel (gray) images are supported. The information about the image has to be set with the setSrcImgInfo. The channel order has to be set to CO_GRAY and the numChannelsUsedForCalculation parameter has to be set to 1. In that case only IMG_OUT_GRAY as rectified image is available.

8.1.2 3-Channel Image

As input format, a "raw" format is used to reduce the overhead. We expect the pixel order to be BGR, RGB, RGBA or BGRA with one byte per channel for each pixel for 8-bit interleaved data storage. In addition plane based storage for BGR and RGB channel order is accepted too.
The functions to set the source image information are setSrcImgInfo and setSrcImgChannelOrder.

The following example shows how the data is positioned in a typical source image with interleaved BGR-order. The data starts with the upper left pixel of the image and is scanned line by line from left to right. Every pixel has 3 channels, and each channel has one byte.

Pixel (N-1)			Pixel (N)			Pixel (N+1)			…
B	G	R	B	G	R	B	G	R	…

In another example the data is positioned in a plane based BGR-order where each channel is stored in a separate data block, which allows faster per channel operations.

Pixel (N-1)	Pixel (N)	Pixel (N+1)	…
B	B	B	…
G	G	G	…
R	R	R	…

8.1.3 Select Single Channel of 3-Channel Image for Processing

The parameters numChannelsUsedForCalculation and intensityChannelUsed of the configuration can be adjusted to use all three channels or a single color channel for the calculation.

numChannelsUsedForCalculation = 3 intensityChannelUsed will not be considered	The image data from all three channels will be processed. If there exist an alpha channel, the alpha channel will be ignored

numChannelsUsedForCalculation = 1
intensityChannelUsed = 0, 1 or 2
	The image data from the first, second or third channel will be processed.

8.2 Destination Image Format

Image type	Buffer-type	Description
IMG_OUT_DISP	unsigned short	The output image is a 16-bit height map image. Each pixel is a 16-bit height value that has a linear relation to the distance between camera and object-surface.
IMG_OUT_DISP_8BIT	unsigned char	Like IMG_OUT_DISP but an 8-bit height map image.
IMG_OUT_RGB	unsigned char	Output image is a rectified RGB image. One unsigned char per pixel and channel with an order of RGB.
IMG_OUT_BGR	unsigned char	Output image is a rectified BGR image. One unsigned char per pixel and channel with an order of BGR.
IMG_OUT_RGBA	unsigned char	Output image is a rectified RGBA image. One unsigned char per pixel and channel with an order of RGBA.
IMG_OUT_BGRA	unsigned char	Output image is a rectified BGRA image. One unsigned char per pixel and channel with an order of BGRA.
IMG_OUT_BGR2	unsigned char	Output image is the second rectified BGR image. One unsigned char per pixel and channel with an order of BGR.
IMG_OUT_GRAY	unsigned char	Output image is a rectified single channel (gray) image. One unsigned char per pixel and channel.
IMG_OUT_GRAY2	unsigned char	Output image is the second rectified single channel (gray) image. One unsigned char per pixel and channel.
IMG_OUT_P3D	float	Output image is an array of 3D points. For each camera pixel (PX,PY) the corresponding coordinate in 3D-space (X,Y,Z) with 32-bit accuracy is given
IMG_OUT_CONF_8BIT	unsigned char	Output image is an 8-bit confidence map image. Each pixel is a confidence value scaled to 8-bit numerical range [0,255] which provides a measure of the likelihood for the height calculation at this pixel position to be correct.
IMG_OUT_CONF_FLOAT	float	Output image is a 32-bit float confidence map image. Each pixel is a confidence value between [0, 1] which provides a measure of the likelihood for the height calculation at this pixel position to be correct.
IMG_OUT_RGB2	unsigned char	Output image is the second rectified RGB image. One unsigned char per pixel and channel with an order of RGB.
IMG_OUT_BGR_PLANES	unsigned char	Output image is a rectified BGR image in plane based channel order. One unsigned char per pixel and channel with an order of BGR (planes).
IMG_OUT_RGB_PLANES	unsigned char	Output image is a rectified RGB image in plane based channel order. One unsigned char per pixel and

		channel with an order of RGB (planes).
IMG_OUT_BGR2_PLANES	unsigned char	Output image is the second BGR image in plane based channel order. One unsigned char per pixel and channel with an order of BGR (planes).
IMG_OUT_RGB2_PLANES	unsigned char	Output image is the second RGB image in plane based channel order. One unsigned char per pixel and channel with an order of RGB (planes).

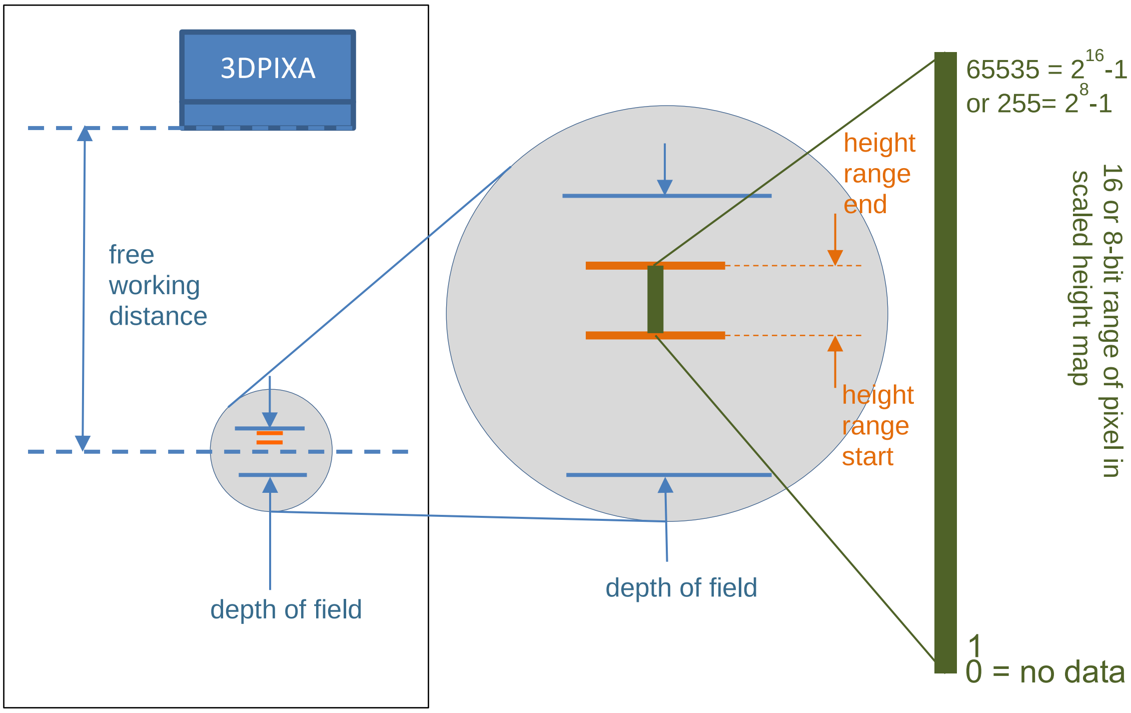
8.2.1 Rectified Image - IMG_OUT_R*/ IMG_OUT_B*/ IMG_OUT_GRAY

The raw image(s) taken from the camera are preprocessed before the height calculation is done. The rectified image is the preprocessed image of camera A that matches the height map image and the point cloud in terms of point positions. So the rectified image at pixel position (x1, y1) holds the color information for the height or the point at the same position of the corresponding result. The output is a 1-dimensional array of unsigned chars, in which every char represents a color channel. (e.g. IMG_OUT_BGR means 3 channels in sequence B, G, R. IMG_OUT_RGBA has 4 channels while IMG_OUT_GRAY has only one channel. The default image type of a rectified image is IMG_OUT_BGR.

Pixel (N-1)			Pixel (N)			Pixel(N+1)			…
B	G	R	B	G	R	B	G	R	…
uchar	uchar	uchar	uchar	uchar	uchar	uchar	uchar	uchar	…

8.2.2 8 bit - / 16 bit – Height Map Image - IMG_OUT_DISP / IMG_OUT_DISP_8BIT

The following diagram shows how a height map is built. Within the free working distance (in which a good image can be acquired), a height range can be set to define the processing area. This area will be scaled to 16 bit with 65535 grayscale values or 8 bit with 255 grayscale values. Within the processing area, all height values can be assigned to the corresponding grayscale values. The default image type of a height map image is IMG_OUT_DISP. IMG_OUT_DISP_8BIT will be used to generate an 8 bit height map.

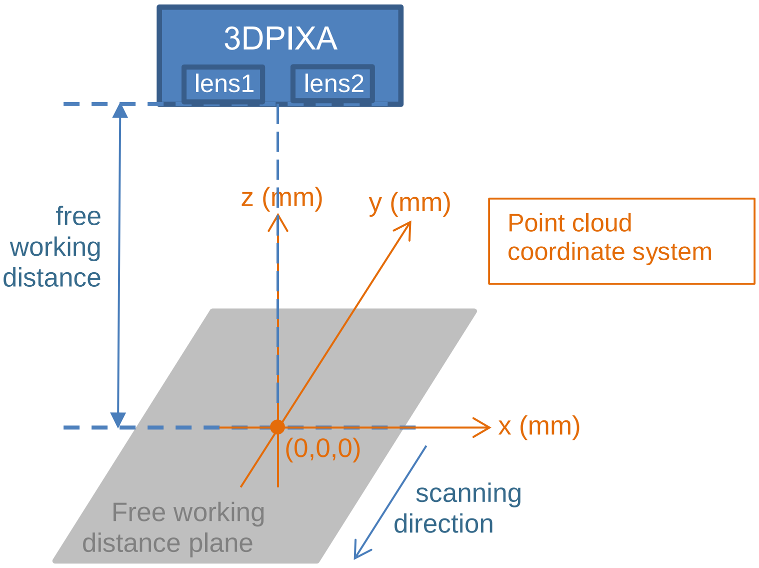


The output is a 1-dimensional array of unsigned chars / unsigned shorts, where there is one grayscale value for each pixel of the rectified output image. The values are ordered row by row from left to right and from top to bottom.

The height in mm can be calculated using the API grayToMm functions. The matching configuration / calibration must be loaded.

Pixel (N-1)	Pixel (N)	Pixel (N+1)	…
unsigned short	unsigned short	unsigned short	…

8.2.3 Point Cloud - IMG_OUT_P3D



The diagram above shows the point cloud world coordinate system. X axis corresponds to point position in mm along sensor direction, y axis corresponds to point position in mm in scanning direction, and z axis corresponds to distance in mm from object surface to the free working distance plane. The origin (0, 0, 0) is placed in the middle between both lenses in the free working distance to the camera. The points in the left part of the image have negative X coordinates, and in the right part they have positive values. In the upper part they have positive Y coordinates and in the lower part they have negative ones. The Z coordinates further away than the free working distance from the camera are negative, and the ones closer are positive. The Y and Z coordinates of the 3D points from the Chromasens HALCON extension have an inverted sign to be compatible with the built-in HALCON functions.

Image type IMG_OUT_P3D will be used to generate a point cloud. The output is a 1-dimensional array of floats. The x-, y-, z-coordinates in the space can also be calculated from the height map image using the grayTo3D (6.5.11) function or from both raw images using the rawImageCoordinatesTo3D (6.5.8).

Point (N-1)			Point (N)			Point (N+1)			…
x-pos	y-pos	z-pos	x-pos	y-pos	z-pos	x-pos	y-pos	z-pos	…
float	float	float	float	float	float	float	float	float	…

8.2.4 8 bit - / float – Confidence Map Image - IMG_OUT_CONF_8BIT / IMG_OUT_CONF_FLOAT

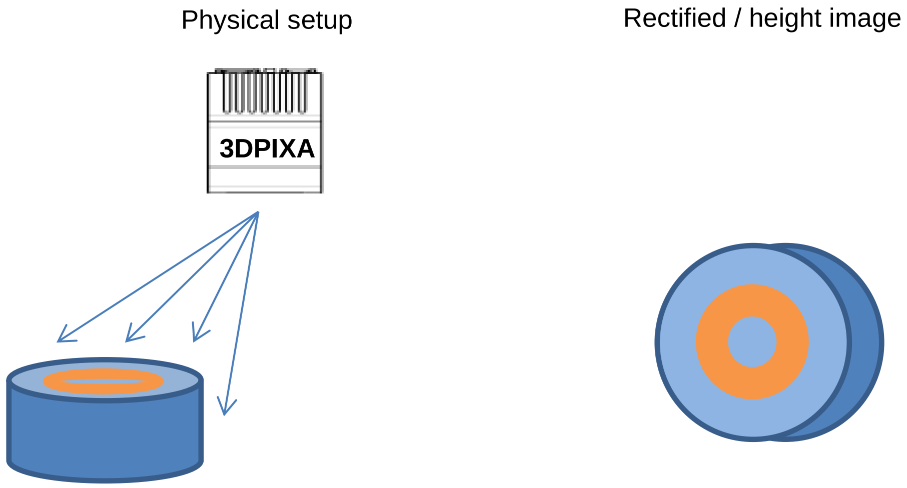
Confidence map provides a measure of the likelihood at each pixel position for the height calculation to be correct. Image type IMG_OUT_CONF_FLOAT is used to generate a 32-bit float confidence map image, where each pixel value is in range [0, 1]. Image type IMG_OUT_CONF_8BIT is used to generate an 8-bit confidence map image, where each pixel value is scaled to range [0, 255]. Larger value in the confidence map indicates that the corresponding height calculation in the height map and point cloud is more confident, and vice versa. Parameter doCalcConfMap in config must be set to 1 in order to output 8-bit/float confidence map. The correlation factor, parameter “minBadKkf” 10.2.3, is the minimum confidence required to calculate an height value. Areas in the confidence and the height map with an confidence below the correlation factor threshold are set to ‘0’.

8.3 Central View / Perspective Correction

With Version 2.3 of the CS-3D-API the central view feature is introduced to the API in a BETA state. This feature corrects the perspective distortion that is generated because of the viewing angles of both cameras. It is already fully working with rawImageCoordinatesTo3D function. If the feature is used with the dense output of points using the getLastImage function there can be some artifacts that will be removed in the future.

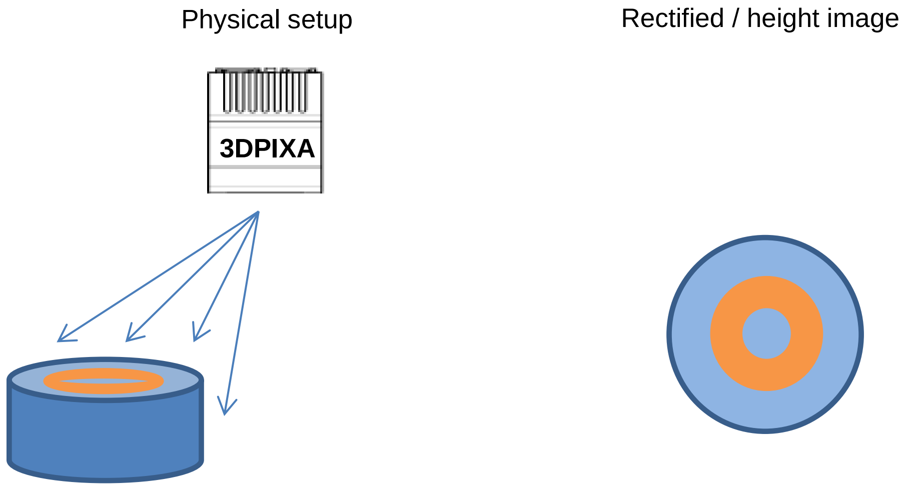
This feature can be activated using the “enableCombinedView = 1” configuration parameter. Also the calibration of the 3DPIXA that is used has to be at least version 2.1

Without perspective correction / central view


Physical setup

Rectified / height image

With perspective correction / central view


Physical setup

Rectified / height image

9 Image Size

With the 3DPIXA we ship a calibration file and a configuration file. The width of the images that can be processed by the CS-3D-Api is fixed for the specific camera. So always the complete image in width has to be passed to the API, while the image height can be flexible.

If you are interested in a small part of the image, please use the ROI functions 6.4.14 to 6.4.18 .

The height of the images is only restricted by the used frame grabber and/or the available amount of PC/GPU RAM. The height of the image has to be set through the function 6.4.1. For more information about the configuration please refer to chapter 10.

10 Configuration

A configuration and a calibration file are shipped with our cameras. The system specific designed calibration-file should not be touched without explicit demand.

The configuration file contains a set of default parameters as initial setting. These parameters can be adjusted to user-specific needs, e.g. file paths, through the API or with any text-editor. Some parameters are not listed in the initial configuration file, which the user can access and adjust (if writable) via API. The default windows-ini format is used for compatibility reasons.

The configuration file can be in any place on the hard-disk, but the path to the calibration file has to be adjusted. If the calibration file is in the same folder like the configuration file setting the “calibrationFile”-parameter to the filename without the path is sufficient.

10.1 Configuration via API

To alter the configuration via API, you first have to get a copy of the active config3DApi-Object with the I3DApi::getConfig (6.3.1) or I3DApi::getActiveConfig (6.3.5) function. Then you can directly access different parameters through the object. For each parameter there is a description in Section 10.2. The manipulated configuration object can now be used with the initialization function 6.2.1 or set as the new configuration with I3DApi::setConfig function (6.3.2) followed by a call of I3DApi::reinitialize (6.2.2). After that the configuration can be saved to a given file.

Starting with version 2.4a, a new configuration parameter called “enableDynamicConfiguration” has been introduced that allows changes to some disparity calculation parameters without the need to reinitialize the API. To activate this feature, set the “enableDynamicConfiguration” parameter (or add it if it doesn’t exist yet) in your configuration file to “1”. Afterwards get a copy of the configuration like before, change parameters as desired and set this configuration object via the I3DApi::setConfig function (6.3.2). In this case, initialization is still required but will only apply calculation parameters that were changed – see also the “enableDynamicConfiguration” section in chapter 10.2.2 Control parameters for a list of possible parameters.

Usage of the “enableDynamicConfiguration” parameter is recommended in situations where calculation parameters need to be changed while the 3D-API is running.

10.2 Parameter Descriptions

10.2.1 General Parameters

These parameters show general setting of the API calculation, calibration related or camera specific information. Most parameters cannot be modified.

Parameter	Example value	Access	Description
calibrationFile	calibration.ini	R/W	Path and filename of provided calibration file
configFile	“C:\data\config.ini”	R	Path and filename of the currently loaded configuration file
dataSource	DATASOURCE_DISK	R	This parameter shows from where the current configuration file is loaded (i.e. from disk or from camera). The possible values are DATASOURCE_DISK and DATASOURCE_CAM
cameraType	CP000470-C01-015-0040	R	Type (product ID) of the camera
numCams	1	R	Number of physical cameras used.

			1, for single camera system 2, for dual camera system
imgWidth	3648	R	Image width of sensor of one (virtual) camera Unit is px.
imgHeight	4000	R	Image height of sensor. Unit is px
offsetImageA	0	R	The offset where the Api begins interpreting the source image as an image of camera A. Unit is px.
offsetImageB	3648	R	The offset where the Api begins interpreting the source image as an image of camera B. Unit is px.
workingDistance	199.8	R	Calibrated working distance of this camera type.
freeWorkingDistance	99.6	R	Calibrated free working distance of this camera type.
resolutionX	0.015	R	Resolution of the system in X direction. Unit is mm per px.
resolutionY	0.015	R	Resolution of the system in transport direction. Unit is mm per px.
serialnumber	207	R	Serial number of the sensor
calibrationVersion	2	R	Version of the calibration file
heightRangeLimitInPx	[-60,60]	R	The height range within which the 3D calculation is very accurate. When parameter enableHeightRangeLimit is set to 1, this limit is applied, and when dStart or dEnd is beyond this limit, it will be automatically adjusted within the limit and a warning will be issued, or an error will be reported. When parameter enableHeightRangeLimit is set to 0, this limit is ignored.
calibHeightRangeLimitInPx	[-120,120]	R	The height range within which the 3D calculation is possible, but the accuracy can be lower if the object surface is scanned outside the height range defined by heightRangeLimitInPx. This limit cannot be switched off.
numWindowTypes	11	R	Number of possible window sizes
cudaDllName[0]	cudaRDT27x27.dll	R	Name and path of dll for each window type. Eg. the window size of this dll is 27x27 pixels in size
cudaDllName[1]	cudaRDT27x3.dll	R	
cudaDllName[2]	cudaRDT15x15.dll	R	
cudaDllName[3]	cudaRDT11x3.dll	R	
cudaDllName[4]	cudaRDT9x9.dll	R	
cudaDllName[…]	…	R	

10.2.2 Control parameters

These parameters control the behavior of the 3D calculation, and correspond to the user image acquisition settings and calculation requirements.

Parameter	Example value	Access	Description
doCalc3DPoints	1	R/W	Set to “0” if you want to disable the calculation of the 3D point cloud.
doCalcRectifiedImage	1	R/W	Set to “0” if you want to disable the calculation of the rectified/texture image. This rectified/texture image is calculated from image captured by camera A.
doCalcRectifiedImageB	0	R/W	Set to “1” if you want to have the second rectified/texture image from camera B.
doCalcConfMap	0	R/W	Set to “1” if you want to enable the calculation of the 8-bit or float confidence map image.
doCalcHeightImage	1	R/W	Set to “0” if you want to disable the calculation of the height image. This also disables the calculation of the point cloud.
enableCombinedView	0	R/W	Set to “1” if you want to enable the central view / perspective correction feature. To enable this feature, the calibration version needs to be at least 2.1. (Chapter 8.3)
numChannelsUsedForCalculation	3	R/W	Number of channels of the image used for calculation. Currently only 1 or 3 are valid options. If an alpha-channel is present it is always ignored. (Chapter 8.1.3)
intensityChannelUsed	0	R/W	With this parameter users can choose which channel of the image (if it is RGB/BGR) is used for the calculation of the height information. Only used if numChannelsUsedForCalculation is set to 1.
switchToDemoModeOnAuthError	0	R/W	Set to “1” if you want to enable to automatically switch to demo mode if no dongle is present.
invertTransportDirection	0	R/W	If the image is acquired in the direction opposite to the defined transport direction, this parameter has to be set to “1”.
numGpus	1	R/W	Maximum number of GPUs used.
debug	0	R/W	To enable debug output set debug >0. in config file this parameter is named “debug”. To view the debug messages on Windows use “DebugView” from http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx On Linux, all log output currently appears in the console window.
useGpus[0]	0	R/W	For each used GPU the physical GPU number has to be assigned. Usually the default values are ok.
useGpus[1]	1	R/W	

saveGpuRam	0	R/W	When set to 1 it reduces the GPU memory usage. This is useful when running multiple instances of the 3D-API while all of them share the same GPU hardware. The default value is 0.
enableHeightRangeLimit	1	R/W	When set to 1, heightRangeLimitInPx is applied, and when dStart or dEnd is beyond this limit, it will be automatically adjusted to the limit, or an error will be reported. heightRangeLimitInPx defines the height range within which the 3D calculation is very accurate. Set this parameter to “0” if you want to use a larger height range up to calibHeightRangeLimitInPx. In this case, the calculation accuracy can lower if the object surface is scanned outside the height range defined by heightRangeLimitInPx.
enableDynamicConfiguration	0	R/W	When set to 1, this enables the option to allow partial dynamic configuration changes without the need to reinitialize the 3D-API. The dynamic configuration changes are only supported by a subset of calculation specific parameters, that are listed below: • dStart and dEnd • brightR and darkR • minStdA • minBadKkf • consistent See chapter 10.2.3 Calculation Specific Parameters for more information about these parameters.
matchingMethod	BLOCK_MATCHING	R/W	Possible Values are • BLOCK_MATCHING • SEMI_GLOBAL_MATCHING This parameter allows to switch the 3D reconstruction method between block matching and SGM. More information on the Semi-Global Matching method can be found in the “Technical-note_CS-3D-API Semi-Global Matching_R1.pdf” document.
heightRangeMode	DYNAMIC_RANGE	R/W	This parameter is used only if the “SEMI_GLOBAL_MATCHING” matching method is active Possible Values are • FIXED_RANGE • DYNAMIC_RANGE With fixed height range, the internal disparity search range is fixed to 128, starting from dStart, and the output height map is truncated by dEnd. Thus, the maximum height search range is 128 pixels. If the input height range (dEnd-dStart) is larger than 128, input dEnd is ignored and set to dStart+128

			automatically. With dynamic height range, any height range is accepted for calculation, and height range is dEnd - dStart.
p1	7	R/W	Penalty value for small disparity change (i.e. 1 pixel disparity). Using a lower penalty for small changes permits an adaptation to slanted or curved surfaces. p1 is in range between 0 and 31. Default value is 7.
p2	100	R/W	Penalty for all larger disparity changes (i.e. larger than 1 pixel disparity). Penalizes discontinuities in height. Such discontinuities are often visible as intensity changes. It always has to be ensured that p2 ≥ p1. p2 is in range between 0 and 127. Default value is 100.
doCameraInfoBlockChecks	0	R	When set to 1, the API performs some additional checks for the correctness of the provided input images. The API checks are performed based on the information from image info-blocks. The API checks that, • Input image(s) and the calibration file are related • In Dual camera setup, the master-slave image pair are related to each other. • In Dual camera setup, the master-slave image pair is given in right order • If one of the input images is vertically inverted • If all the input image lines are present Please note that, in-order to perform these checks the first line info block and each line info blocks should be present in the input images.

10.2.3 Calculation Specific Parameters

These parameters specify calculation parameters and thresholds, and can influence the quality of the results. For further information especially about the adjustment of the parameters, please refer to the “CS-3D-Viewer Manual” chapter 6.

Parameter	Description	Default value	Unit
windowType	Index of window that is used for the 3D calculation. A List of all sizes can be found in chapter 10.2.3.1	0	[0,1, .., 10]
brightR	Minimum gray level for height map calculation	3.0	[gray level]

darkR	Maximum gray level for height map calculation	254.0	[gray level]
minStdA	Minimum contrast (standard deviation)	0.3	[without a unit]
minBadKkf	Minimum correlation coefficient accepted	0.25	[without a unit]
consistent	Maximum difference between LR and RL matching	1	[px]
dStart	Lower height range limit (search start)	-50	[px]
dEnd	Upper height range limit (search end)	50	[px]
dY	Vertical displacement	0.0	[lines]
dispThreshErr	Smallest height value	0.1	[without a unit]
heightResolutionReduction	When set to 1, 3D calculation will speed up, at the cost of certain degree of height resolution reduction. This can be an efficient way to increase the calculation speed when the reduced height resolution still fulfills the requirement of your application.	0	[without a unit]

10.2.3.1 Window Types

Number	Name	Width	Height
0	27x27	27	27
1	27x3	27	3
2	15x15	15	15
3	11x3	11	3
4	9x9	9	9
5	7x7	7	7
6	5x5	5	5
7	3x3	3	3
8	11x11	11	11
9	13x13	13	13
10	21x21	21	21
11	7x27	7	27
12	49x49	49	49
13	99x99	99	99

10.3 System Parameters

For a detailed description of the system parameters and how to adjust them, please refer to “CS-3D-Viewer Manual” chapter 7.

11 Using the API

Generally speaking, the images can be processed in parallel or sequentially. The diagrams below show the parallel processing scheme and sequential processing scheme respectively. In parallel processing, the threads of image acquisition and calculation are independent of each other. In sequential processing, the next image(s) will be acquired only if the processing of the previous image(s) is finished.

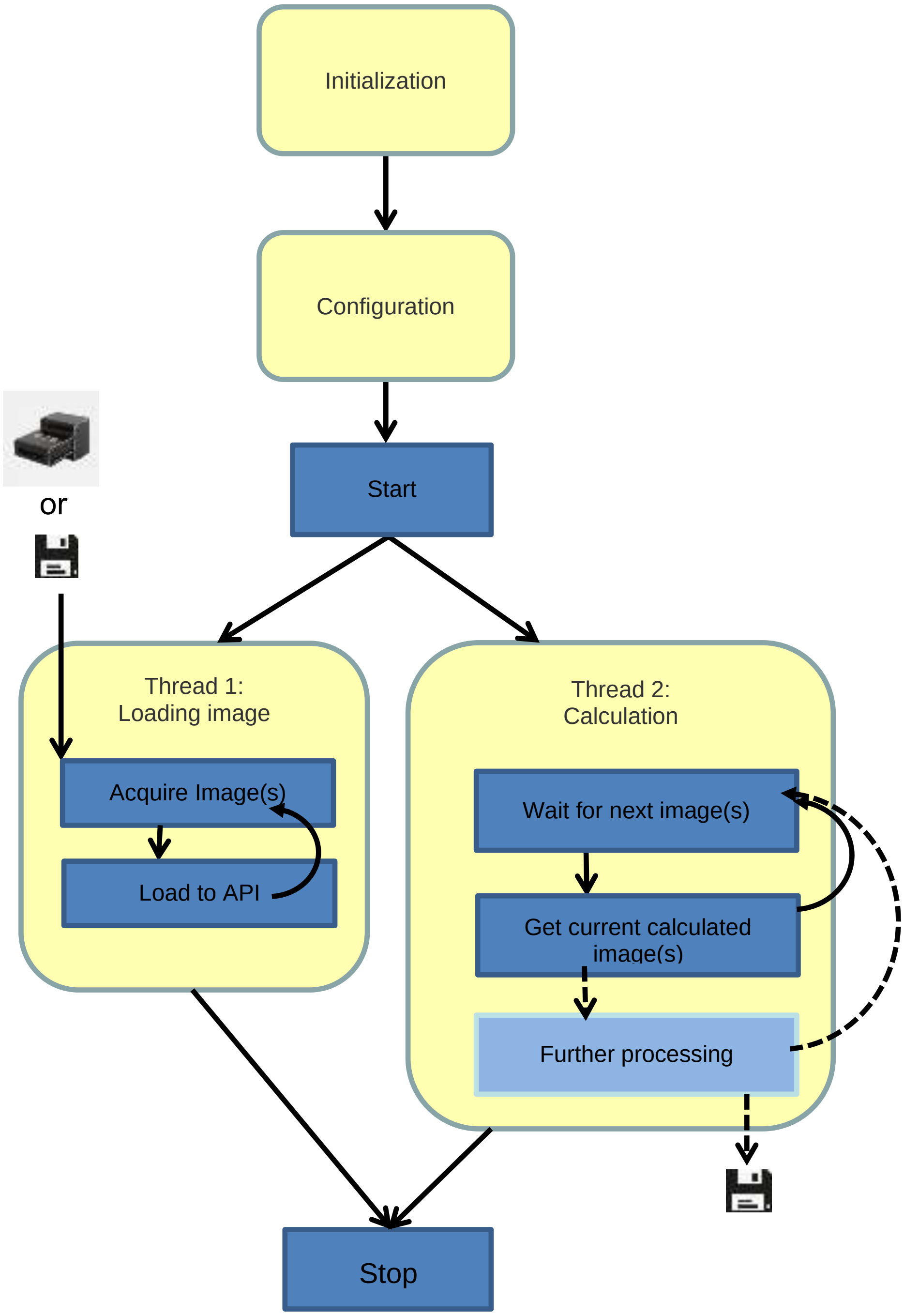


Figure 2: parallel image processing with API

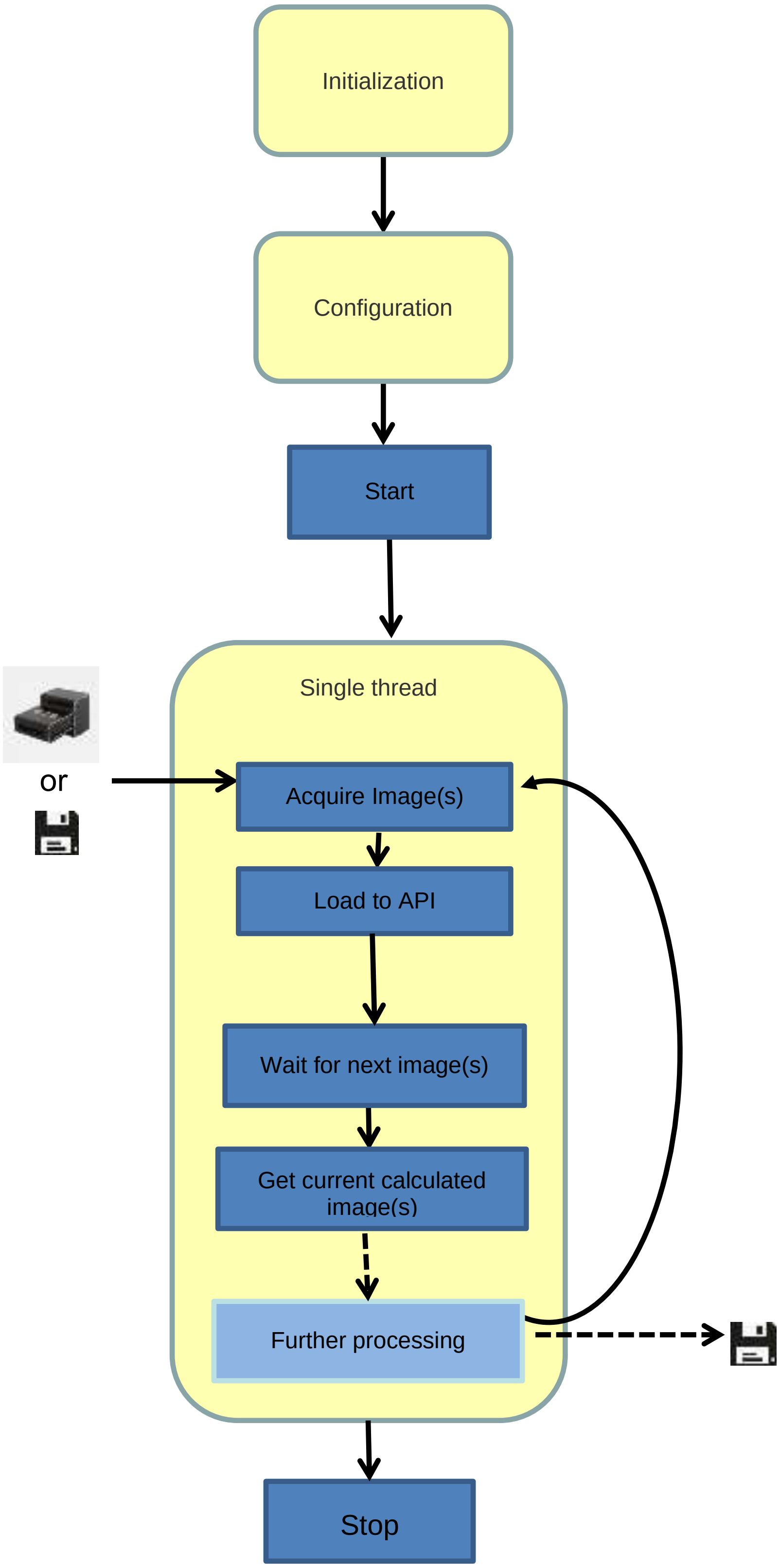


Figure 3: sequential image processing with API

11.1 Initialization

First we focus on the initialization of the API and the handling of the configuration. The Chromasens 3D API uses the CS3D namespace. After loading the shared library and executing the API creator function “CS3DApiCreate”, we can access the whole collection of functions of the API through this object. An initial configuration file can be passed on creation or loaded afterwards using the loadConfig function.

It is important to know that if you alter the configuration and want the changes to become active, you have to set the new configuration via the I3DApi::setConfig function and call the I3DApi::reinitialize function after that. Or you can do an initialization of the system directly via I3DApi::initialize (&cfg) and pass a reference of the configuration object. Before (re-)initialization, the calculation has to be in the stopped state.

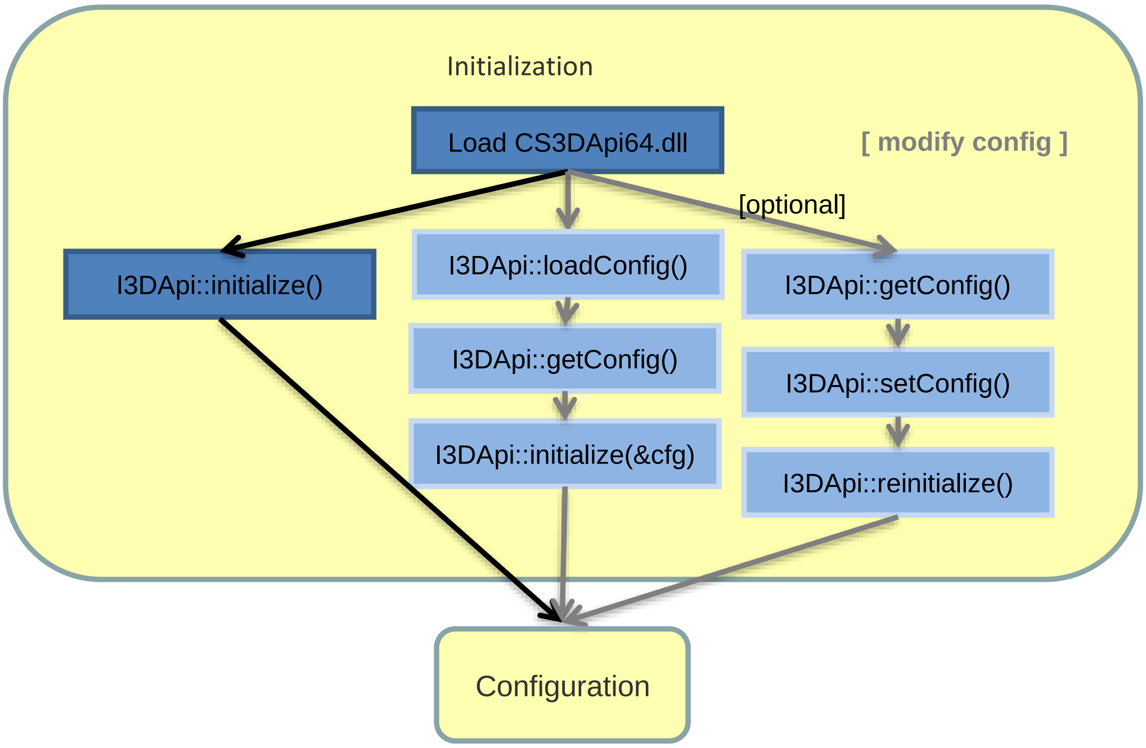


Figure 4: Initialization Process

Initially, we must provide the API with image dimensions, number of channels, size of the source image, as well as the bits per pixel and the line pitch.

Calling the start function starts the calculation. If no input image is set, the calculation threads are idling. The calculation can be stopped using the stop function. With the I3DApi::stopBlocking() function, the calculation stops before the actual requested calculation is finished

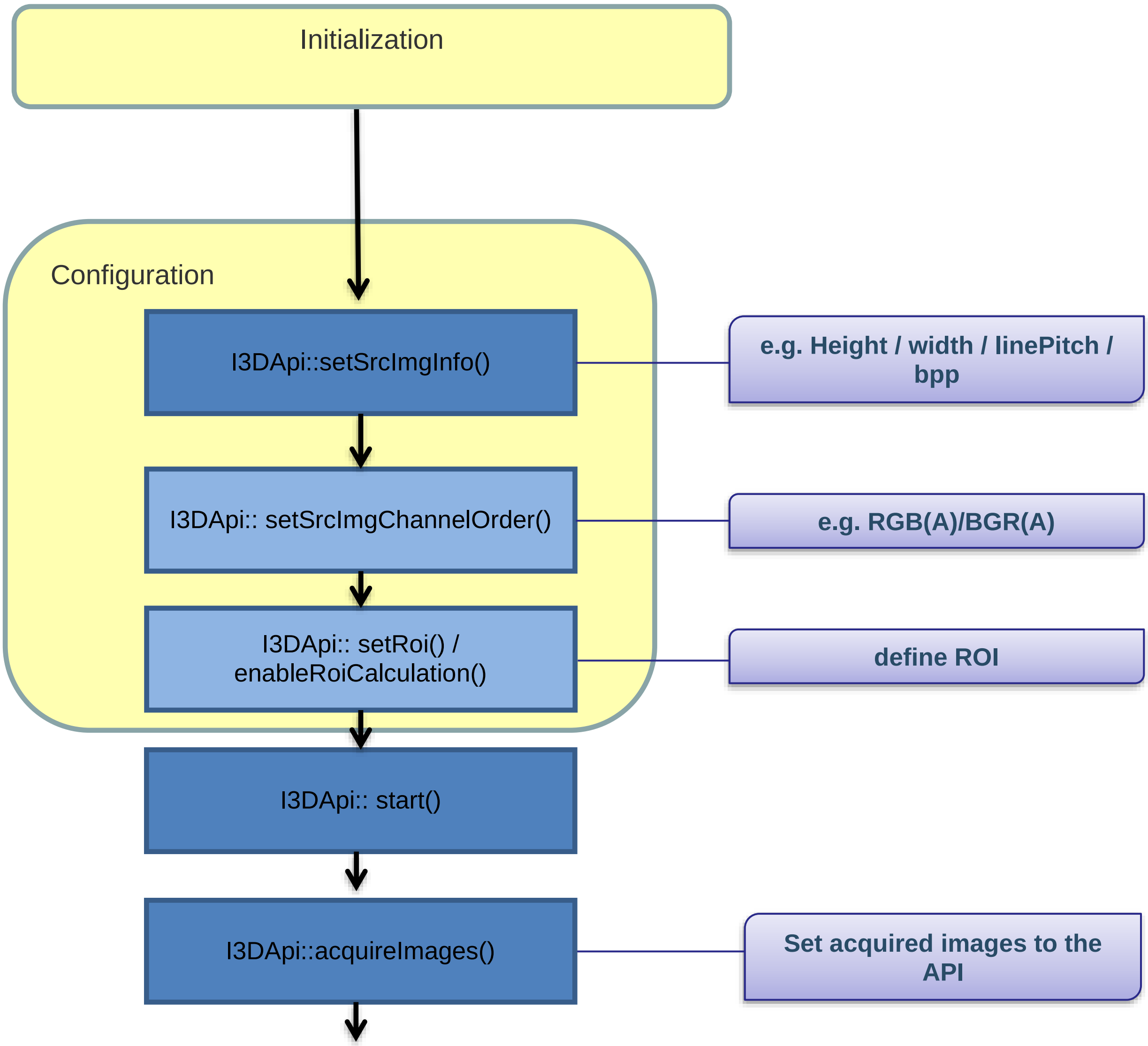

Initialization
Configuration
I3DApi::setSrcImgInfo()
e.g. Height / width / linePitch / bpp
I3DApi:: setSrcImgChannelOrder()
e.g. RGB(A)/BGR(A)
I3DApi:: setRoi() / enableRoiCalculation()
define ROI
I3DApi:: start()
I3DApi::acquireImages()
Set acquired images to the API

Figure 5: Initialization Process

11.1.1 Example

```
#include <afxwin.h>

#include "..\helpers\helper.h"
#include "..\..\..\3dapi\includes\CS3DApiDll.h"

#include <opencv\cv.h>
#include <opencv\highgui.h>

#include <iostream>

#if _DEBUG
#pragma comment(lib,"opencv_core248d.lib")
#pragma comment(lib,"opencv_highgui248d.lib")
#else
#pragma comment(lib,"opencv_core248.lib")
#pragma comment(lib,"opencv_highgui248.lib")
#endif
```

```
using namespace std;
using namespace CS3D;

I3DApi * m_p3dapi=NULL;

int const numCams=2;
int const numImgs=1;

int _tmain(int argc, wchar_t* argv[]) {

       int widthIn0,heightIn0,channelsIn0,bpp0,linePitch0;
       int widthIn1,heightIn1,channelsIn1,bpp1,linePitch1;
       long sizeIn0,sizeIn1;
       unsigned char* buffer[2];

       cv::Mat inImg[numCams];


       // Load 3D-Api Dll
       accessDllObj pAPI(&m_p3dapi,L"C:\\Program
       Files\\Chromasens\\3D\\dlls\\CS3DApi64.dll",
       "CS3DApiCreate", "C:\\Users\\Public\\Documents\\Chromasens\\3D\\sample
       images\\two camera system\\config.ini",0);

       //check if dll is loaded
       if (m_p3dapi==0){
              printf ("Error loading dll\n");
              return -1;
       }

       //get copy of active config object
       config3DApi *cfg =m_p3dapi->getConfig();

       //check if config is returned
       if(cfg == NULL){
              printf ("can't get config\n");
              return -2;
       }

       //(optional) adjust config parameters
       //cfg->dStart=-50;
       //cfg->dEnd=50;

       //after loading dll, initialize the api
       if ((m_p3dapi->initialize(cfg))<0){
              printf ("can't init system/load config file\n");
              return -2;
       }

       char camFilename[500];
       char * testImgDir = "C:\\Users\\Public\\Documents\\Chromasens\\3D\\sample
       images\\two camera system\\";

       //load test source images
       for (int i=0;i<numCams;i++){
              sprintf(camFilename,"%s\\%c_%04d.bmp",testImgDir,'A'+i,0);
              inImg[i] = cv::imread(camFilename);
       }

       //check if images were loaded correct
```

```
if ((inImg[0].rows == 0) ||(inImg[1].rows == 0)){
        printf("error loading sample images\n");
        return -3;
}

//set image information
widthIn0=inImg[0].cols;
heightIn0=inImg[0].rows;
channelsIn0=inImg[0].channels();
sizeIn0=widthIn0*heightIn0*channelsIn0*sizeof(char);
bpp0=8*channelsIn0;
linePitch0=inImg[0].step;
buffer[0]=inImg[0].data;

if ((m_p3dapi->setSrcImgInfo(0,widthIn0,
        heightIn0,channelsIn0,bpp0,linePitch0,sizeIn0))<0){
        printf ("image info not acceptable for calculation\n");
        return -4;
}

//set pointer to source image of first camera
if (m_p3dapi->setSrcImgPtr(0,(char*)buffer[0])<0){
        printf("can’t set src pointer\n");
        return -5;
}
```

As input format, a “raw” format is used. For additional format description, please refer to section 8.1. Parameters for the second image source can be set in the same fashion.

```
//now the calculation process is started with
if (m_p3dapi->start()<0){
        printf("3D-API has to be initialized first.\n");
        return -5;
}
```

Now you can load the input images in parallel and copy the output images for further processing

11.2 Loading Process

After initialization of the API we need to provide it with pairs of source images. This is the loading process. Since version 2.5 of the 3D-API there exist two ways for loading acquired images to the API:

1.) Manual approach with a bit more control over the workflow, see Figure 6.

Note: This approach is only available to Windows platforms.

2.) Convenient approach which is easier to use.

Important: Make sure not to mix the two loading process approaches. The more convenient approach 2.) internally executes all steps done in 1.).

11.2.1 Manual Loading Process

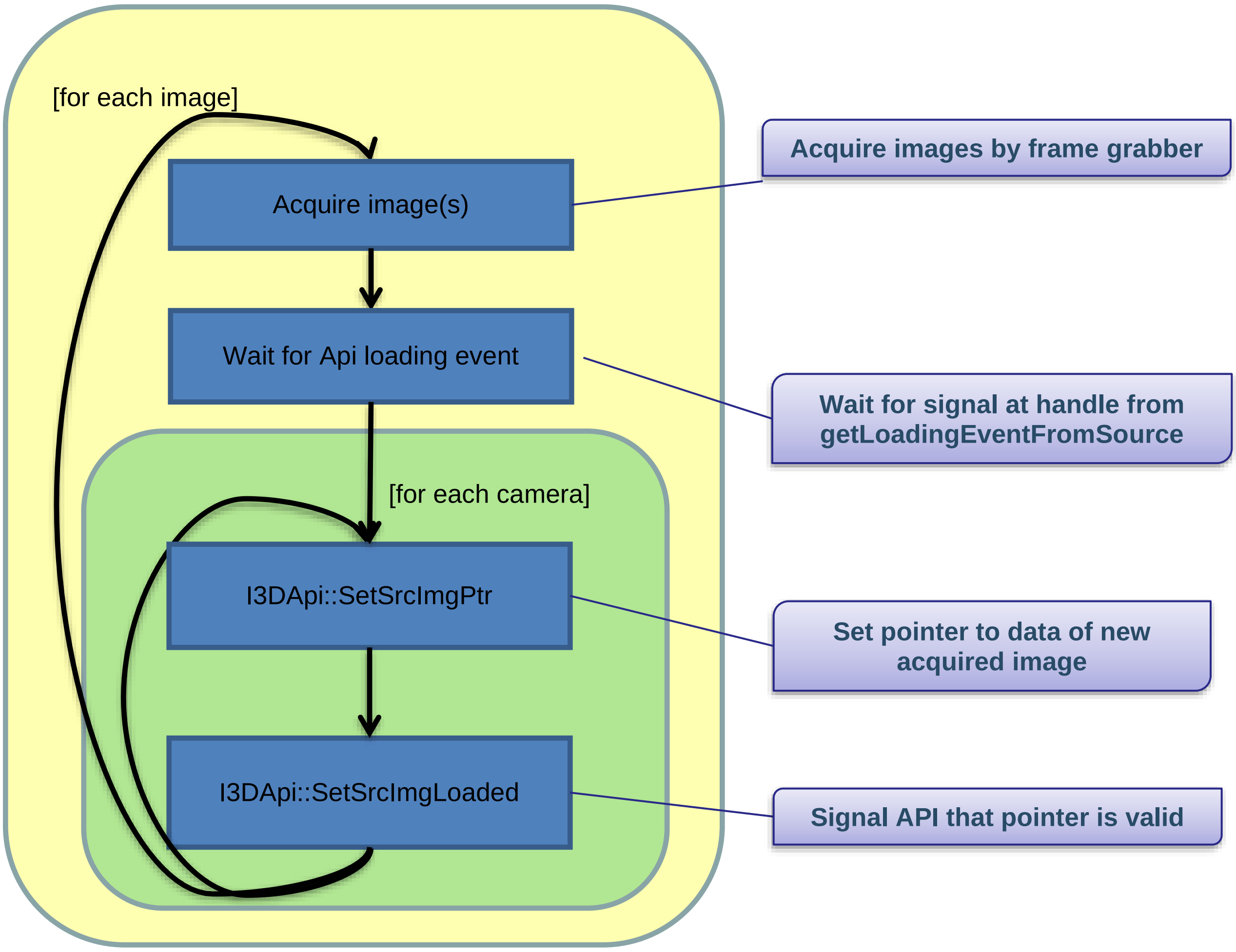

[for each image]
Acquire image(s)
Wait for Api loading event
[for each camera]
I3DApi::SetSrcImgPtr
I3DApi::SetSrcImgLoaded
Acquire images by frame grabber
Wait for signal at handle from getLoadingEventFromSource
Set pointer to data of new acquired image
Signal API that pointer is valid

Figure 6: Manual Loading Process

We first get two event handles which will signal when it is ok to change the pointer to a new source image. After setting the pointer to the new image buffer and/or loading a new image. We signal via the setSrcImgLoaded(camNr) that it is safe for the API to process the image. With this call the loadEvent is also reset, so we can use it again to wait for the images being processed.

11.2.1.1 Example

```
//get handles to check if it is ok to change the source pointer
HANDLE loadEvents[numCams];
for (int c=0;c<numCams;c++){
        loadEvents[c]=m_p3dapi->getLoadingEventForSource(c);
}

for (int j=0;j<numImgs;j++){
        //wait for API
        WaitForMultipleObjects(numCams, loadEvents, true, INFINITE );

        for (int c=0;c<numCams;c++){
                sprintf(camFilename,"%s\\%c_%04d.bmp",testImgDir,'A'+c,j);
                inImg[c] = cv::imread(camFilename);
                //check if images were loaded correct
                if (inImg[c].rows == 0){
                        printf("error loading sample images\n");
                        return -3;
                }

                buffer[c]=inImg[c].data;

                if ((m_p3dapi->setSrcImgPtr(c,(char*)buffer[c]))<0){
                        printf("can’t set src pointer\n");
                        return -5;
                }

                m_p3dapi->setSrcImgLoaded(c);
        }
}
```

11.2.2 Convenience Loading Process

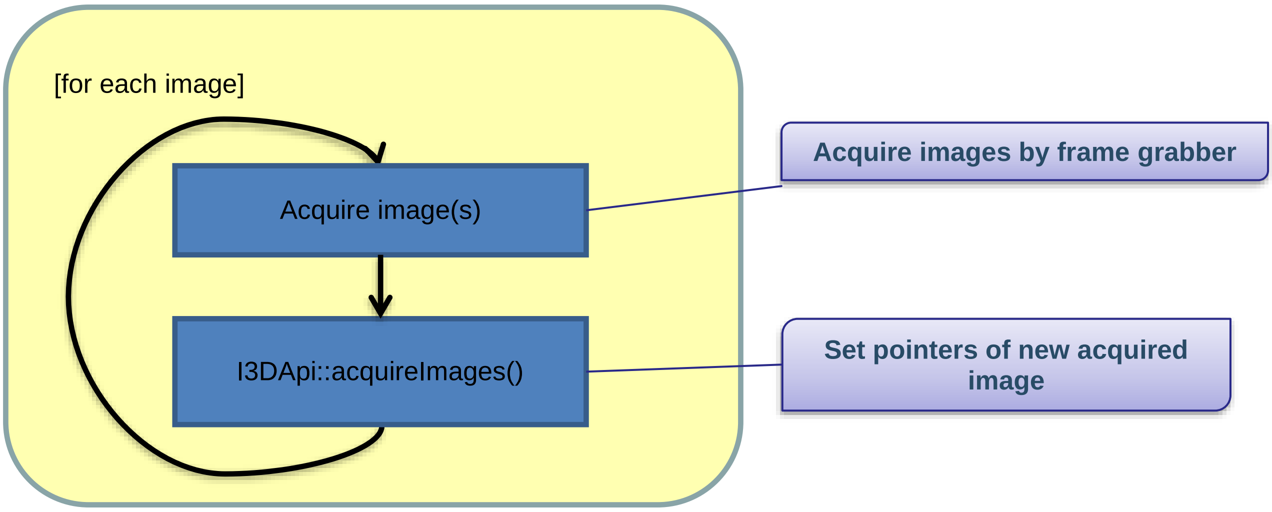

[for each image]
Acquire image(s)
I3DApi::acquireImages()
Acquire images by frame grabber
Set pointers of new acquired image

Figure 7: Convenient Loading Process

In this example the images should also be acquired by frame grabber. The acquired images can then be set to the API using the I3DApi::acquireImages() function. I3DApi::acquireImages() will wait internally until it is allowed to set images to the API.

11.2.2.1 Example

```
for (int j=0;j<numImgs;j++){
        for (int c=0;c<numCams;c++){
                // Loading of images either from disk of frame grabber
                //…
                buffer[c] = […] // Set the pointers for each acquire image
                // Loading of images done
        }
        uint32_t imgNr = 0;
        // Assuming dual camera, so it sets two pointers.
        // For compact the second one can be NULL
        m_p3dapi->acquireImages(imgNr, buffer[0], buffer[1]);
        //… continue processing
}
```

11.2.3 Loading Mask Image Data

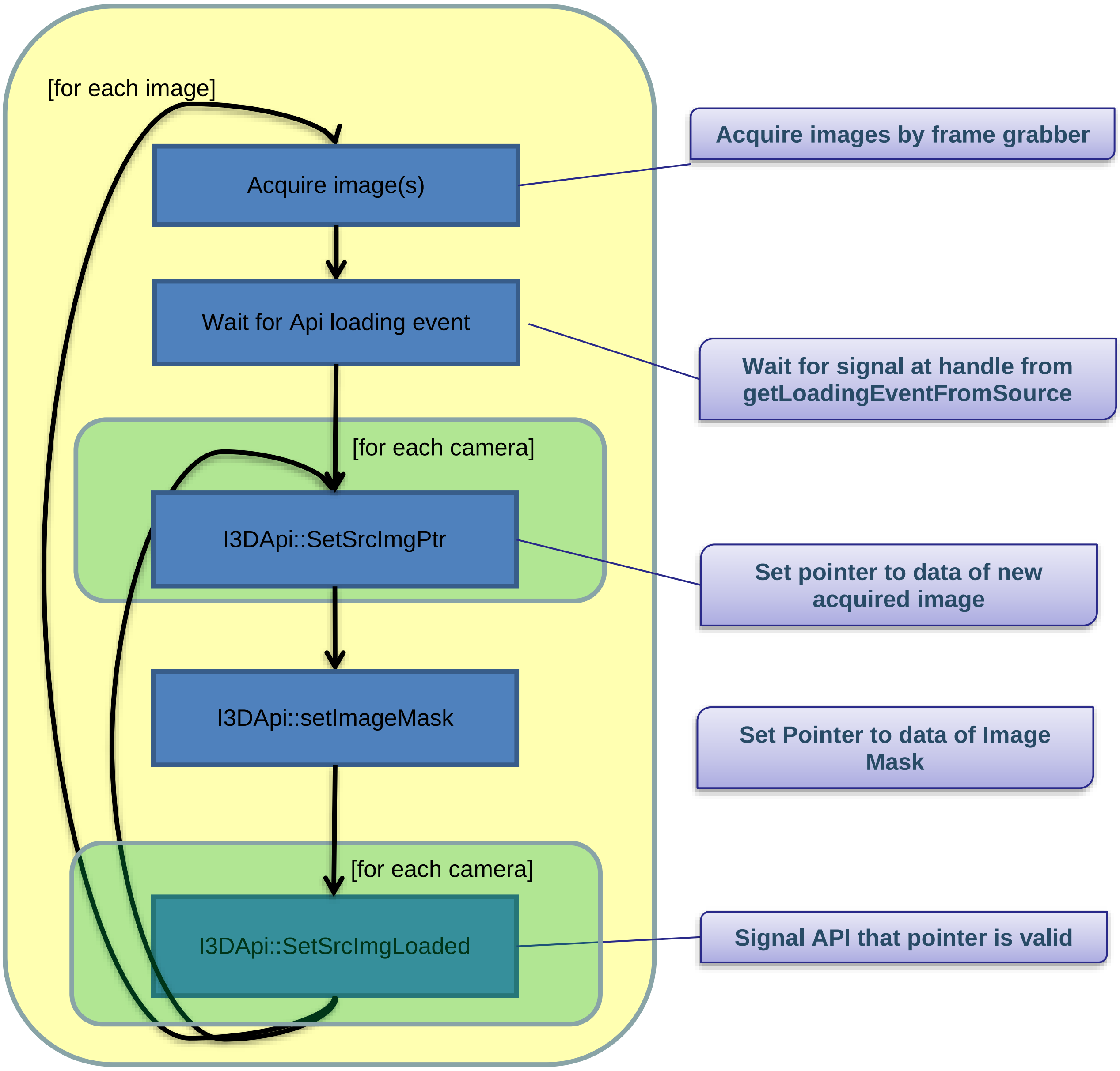

[for each image]
Acquire image(s)
Wait for Api loading event
[for each camera]
I3DApi::SetSrcImgPtr
I3DApi::setImageMask
[for each camera]
I3DApi::SetSrcImgLoaded
Acquire images by frame grabber
Wait for signal at handle from getLoadingEventFromSource
Set pointer to data of new acquired image
Set Pointer to data of Image Mask
Signal API that pointer is valid

Figure 8: Setting Mask in Convenience loading process

Similar to calculating only a required rectangular area of the source image using ROI feature of the API, calculation can be done only on specific pixel regions of the image of any shape. It can be achieved by providing information about required regions in a new, single channel grayscale image to the API. This image is termed as Mask image. In the Mask image, pixels of regions in which calculation is required must be greater than zero. Mask information can be provided to the API using the function “setImageMask()”. Mask information should be provided to the API, after the API is started and before the even handles “setSrcImgLoaded” are set, in case of Manual loading process are used. If convenience loading process are used, Mask data has to be set before the function “acquireImages()” is called. The Mask input should be same size as of the source images.

11.2.3.1 Example

```
for (int j=0;j<numImgs;j++){

	buffer = […] Set the pointers of Mask image
```

```
        m_p3dapi->setImageMask(buffer); Provide Mask information to the API
}
```

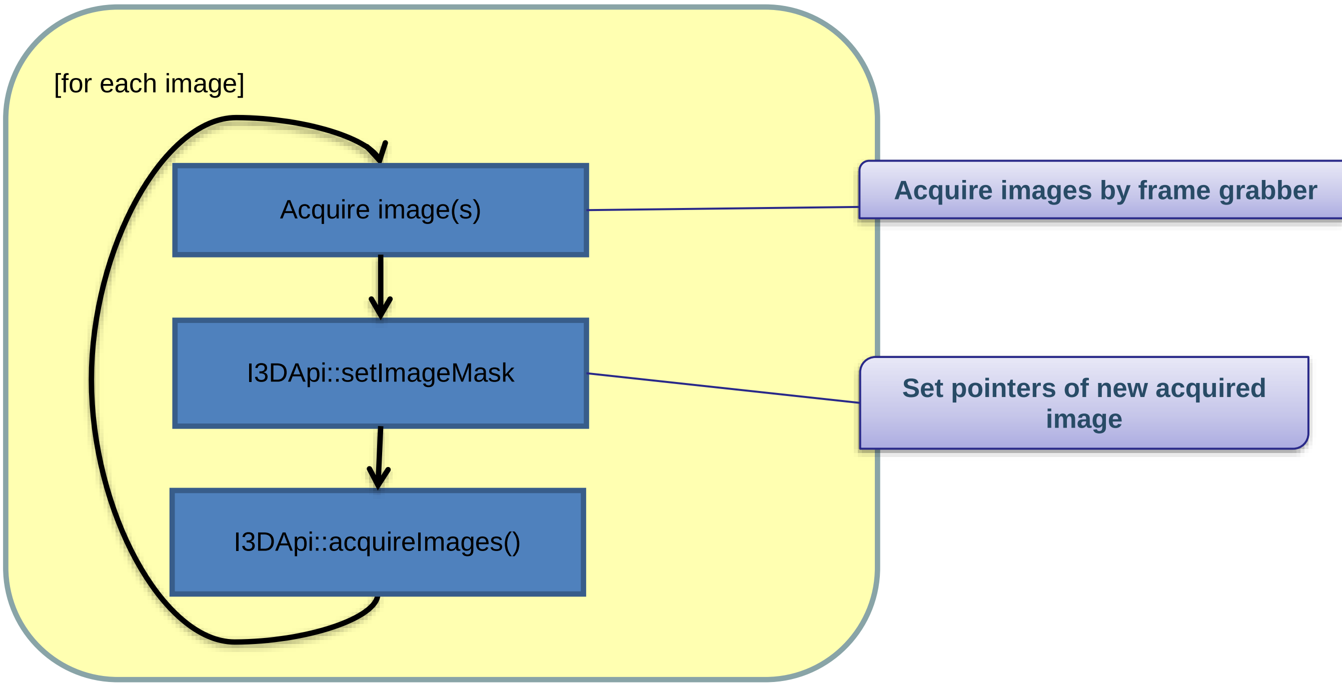


Figure 9: Setting Mask in Convenience loading process

11.2.3.2 Dilate Mask

It is possible to perform calculation on some additional pixels along the borders of the input mask. In this case, API increases the size of mask by performing dilate operation on the Mask image. This feature can be activated by adding the option “dilateWinSize” in C3D-API section of configuration file. The value of the “dilateWinSize” must be the required additional number of pixels in the border. The maximum value of “dilateWinSize” is 99.

11.3 Calculation Process

After the initialization is done and loading of the images is completed, we focus on the calculation process.

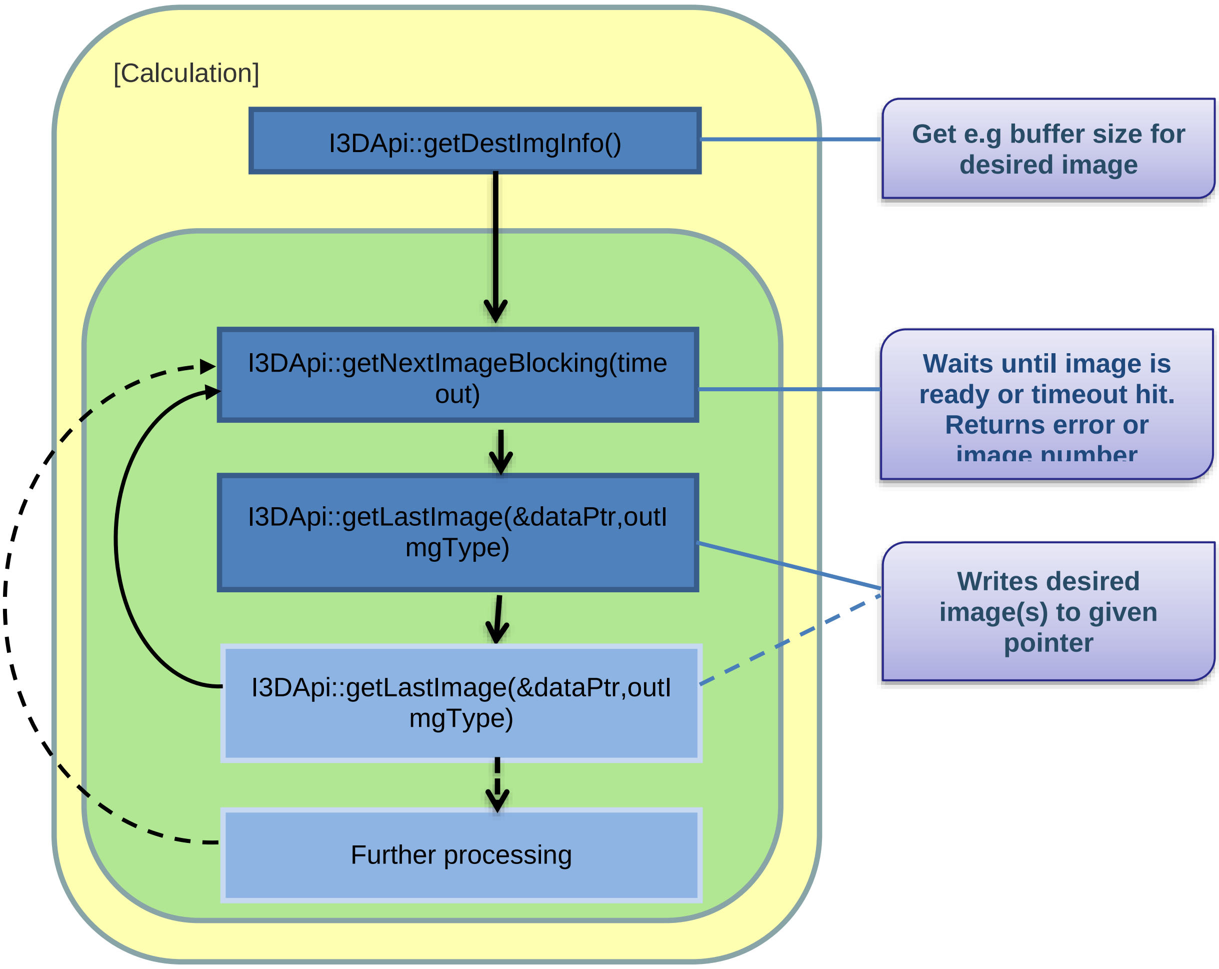
To get the dimensions, channel number and output image size in byte, the getDestImgInfo() function can be used. Then a destination buffer can be allocated to store the calculated image. The getNextImgBlocking() blocks the execution until the next image has been calculated, or a defined timeout is reached. If the next image is ready, the actual image number is returned. The getLastImage() function will provide a pointer to the destination buffer and the desired image type as soon as the copy process to the user-managed destination buffer is finished. Then the image can be saved or further processed by the user. If different types of images are needed by the user, they can be requested then. This must be done before calling the getNextImageBlocking() again or explicitly freeing the API internal output image via the freeDestImage() function.

11.3.1 Example

```
//get information about destination images
m_p3dapi->getDestImgInfo(
        IMG_OUT_DISP,widthOutDisp,heightOutDisp,channelOutDisp,sizeOutDisp);
m_p3dapi->getDestImgInfo(
        IMG_OUT_BGR,widthOutBgr,heightOutBgr,channelOutBgr,sizeOutBgr);
```

```
//allocate memory for height map image
short * dispBuffer=new short[widthOutDisp*heightOutDisp*channelOutDisp];

//allocate memory for bgr image
unsigned char * bgrBuffer=new unsigned char[widthOutBgr*heightOutBgr*channelOutBgr];

//adjust imgDir to a folder where the result images should be written to
char *imgDir= "C:\\Temp";

char *newFileDisp= new char [500];
char *newFileRgb= new char [500];

bool ret=0;
int imgCnt=0;

cv::Mat imgMatDisp(heightOutDisp,widthOutDisp,CV_16U); //needed for 16bit image
cv::Mat imgMatBgr(heightOutBgr,widthOutBgr,CV_8UC3);

for (int i=0;i<numImgs;i++){
       imgCnt=m_p3dapi->getNextImgBlocking(20000);

       if (imgCnt <0){
              i--;
              printf("got timeout, try again ...\n");
              continue;
       }

       if((m_p3dapi->getLastImage(&dispBuffer,IMG_OUT_DISP))<0){
              printf("error getting last height map image\n");
              return -1;
       }

       if((m_p3dapi->getLastImage(&bgrBuffer,IMG_OUT_BGR))<0){
              printf("error getting last bgr image\n");
              return -2;
       }

       printf("Writing images ...\n");

       sprintf(newFileDisp,"%s\\disp_%d.png",imgDir,imgCnt);
       sprintf(newFileRgb,"%s\\rgb_%d.bmp",imgDir,imgCnt);

       short* dispBufferOri=dispBuffer;

       //write height map buffer
       for (int row=0;row<heightOutDisp;row++){
              for (int col=0;col<widthOutDisp;col++){
                     imgMatDisp.at<short>(row,col)=*dispBuffer;
                     dispBuffer++;
              }
       }
       dispBuffer=dispBufferOri;


       //saveDisp
       ret |=cv::imwrite(newFileDisp,imgMatDisp);


       //saveRGB
       imgMatBgr.data=(unsigned char*)bgrBuffer;
       ret |=cv::imwrite(newFileRgb,imgMatBgr);
```

```
        if (ret!=1){
                printf("can't write image %d\n",i);
        }
}
```



[Calculation]
I3DApi::getDestImgInfo()
Get e.g buffer size for desired image
I3DApi::getNextImageBlocking(timeout)
Waits until image is ready or timeout hit. Returns error or image number
I3DApi::getLastImage(&dataPtr,outImgType)
I3DApi::getLastImage(&dataPtr,outImgType)
Writes desired image(s) to given pointer
Further processing

Figure 10: Calculation Process

11.4 *Demo-mode*

The demo-mode can be activated via the setDemoMode functions or automatically by setting the “switchToDemoModeOnAuthError” configuration parameter to “1”. If the parameter is set to “1” the demo-mode is automatically activated if no dongle is present.

In demo-mode the API behaves like in normal mode, except that no “real” calculation is done. Instead of using the set input images and calibration information the API loads precalculated data from files and outputs that to the user. This mode enables the user to explore and even implement the API without having a dongle.

12 Compatible Video Cards

In general every CUDA 3.5 capable video card can be used. Some cards we tested for compatibility.

Compatibility tested:

- NVidia GTX 780 TI
- NVidia GTX 980
- NVidia GTX Titan
- NVidia GTX 980 TI
- NVidia GTX 1060
- NVidia GTX 1080
- NVidia GeForce GTX 1080 Ti

13 Errorcodes

13.1 C++

Error	Code	Description	Possible actions
CS3D_SUCCESS	0	Everything is alright	
CS3D_CUDA_ERROR	-107	CUDA related error occurred.	Check the log file for more detailed information about the error.
CS3D_INVALID_STATE	-106	Calculation in has entered an invalid state.	The calculation process of a work package in semi global matching method has entered an invalid state. Please investigate the debug log file for detailed information about the error.
CS3D_INPUT_IMAGE_TIMEOUT	-105	API unable to calculate the required image results within the given time range.	Increase the timeout value and change the timeout value with respect to the size of the sour images.
CS3D_MASK_IP_CORRUPT	-104	Input mask image data is corrupt.	Mask image is not loaded right. The mask image data should be of a single channel grayscale image. Please load mask image of same size as of the source left image / image A.
CS3D_LICENSE_TOO_OLD	-103	Your license is not valid for the version of the software you are trying to start.	
CS3D_NO_LICENSES_LEFT	-102	Another instance of the API is already running.	
CS3D_TRIAL_PERIOD_OVER	-101	Your trial period is over.	
CS3D_AUTHENTICATION_PROBLEM	-100	No dongle is present, another instance is already running, user has no license for called function, trial period is over.	Double check if you dongle is plugged in and the device "CSUSB Ver 2.0" is recognized by you PC. You can check that via the device-manager. For an extent of the trial period please contact the support.
CS3D_UNDEFINED	-99	Undefined error occurred.	Please contact the support for further instructions.
CS3D_CONFIG_NOT_FOUND	-98	Configuration file not found.	The file that is passed to the function is not found. Double check if the path and the filename are

			correct.
CS3D_CONFIG_CORRUPT_OR_CALIBRATION_NOTFOUND	-97	Important parameters in the configuration file do not exist or are set to improper values.	Verify that the current configuration file is not corrupted. If the error remains, contact the support.
CS3D_CALIBRATION_FILE_NOT_FOUND	-96	Calibration file not found.	The path to the calibration file within the configuration is not correct. Please adjust it.
CS3D_CALIBRATION_FILE_CORRUPT	-95	Important parameters in the calibration file don't exist or are set to improper values.	Verify that the current calibration file is not corrupted. If the error remains, contact the support.
CS3D_CORE_NOT_INITIALISED	-94	System is not initialized. Initialize it before starting the calculation.	Call the initialize/reinitialize function before starting the calculation.
CS3D_CONFIG_WRITE_FAILED	-93	Can't write the configuration file, check permissions.	Check if you have the permission to write to the file.
CS3D_CANT_START_SYSTEM	-92	Can't start the calculation system, initialize it first.	Check if every function you call before returns without error. If not, resolve the other errors first.
CS3D_SYSTEM_ALREADY_STARTED	-91	System is already started.	This is not a hard error. You just tried to start the system twice.
CS3D_SYSTEM_NOT_STARTED	-90	Start system before calculating.	Start the system via the start-function before using it for calculations.
CS3D_CUDADLL_NOT_FOUND	-89	Dll for chosen window size not found, check permissions.	Check if you have read access to the directory you installed the CS-3D-Api and if there are some “cudaRDT*.dll” dlls. If not reinstall the CS-3D-Api.
CS3D_WRONG_IMAGE_SIZE	-88	Set source image information doesn't match parameters of configuration/calibration.	The image is not supported by the Api. Please load the correct configuration / calibration shipped with your Chromasens 3DPIXA. If you load a demo image from the CS-3D-Api please double check if you choose the correct configuration from the same directory.

CS3D_CALC_IMAGE_TIMEOUT	-87	Timeout occurred while waiting for calculation of next image.	Verify that the timeout is long enough so that the GPU has enough time to calculate the results. Also verify that the loaded image can be calculated.
CS3D_NO_OUTPUTIMAGE_READY	-86	There is no image calculated.	Load an image and wait some time before the result is ready.
CS3D_FUNCTION_NA	-85	Function is not available, e.g. deactivated in configuration.	Double check if the function you use is available and not disabled in the configuration.
CS3D_SYSTEM_SHUTDOWN	-84	The system has been shut down.	Don’t shut down the system if you want to use it at a later time.
CS3D_CAM_NA	-83	The chosen camera is not available	Remember that the first camera is camera “0”, the second is “1” and so on. Double check if you have the correct configuration loaded.
CS3D_GPU_NOT_FOUND	-82	The chosen GPU is not available / not CUDA 3.5 capable.	Check the configuration if the correct GPU is chosen. The first GPU in your system is GPU “0”, the second is “1” and so on. Performing the calculation via Remote Desktop Connection is also not possible for some GPUs.
CS3D_NO_IMAGE_INFO_SET	-81	No information about the input image set.	You have to set the image information before starting the calculation.
CS3D_GPU_MEMALLOC_ERROR	-80	Error of GPU memory allocation, perhaps not enough GPU memory.	Check if the GPU has enough physical memory.
CS3D_MEMALLOC_ERROR	-79	Error of GPU or CPU memory allocation.	Check if there is enough physical GPU and PC RAM available.
CS3D_MEMALLOC_UNDEFINED	-78	Other error of memory allocation.	Check if enough physical RAM is available.
CS3D_CUDA_DEVICE_NOT_FOUND	-77	No CUDA capable device found.	Check if you have a CUDA 3.5 capable device installed.
CS3D_CUDA_DEVICE_COMPCAP_TOOLOW	-76	CUDA compute capability of device is too low.	Check if the GPU you have chosen in the configuration is CUDA 3.5 capable.

CS3D_CUDA_DEVICE_NOT_ENOUGH_MEM	-75	Not enough GPU memory.	Check if the GPU has enough physical memory.
CS3D_DEMO_ERROR	-74	Error occurred while in demo mode (e.g. file not found).	Check if you have at least read permission to the folder where you installed the Api. Reinstall the CS-3D-Api to be sure everything is installed correctly.
CS3D_INVALID_CALIBRATION	-73	The calibration information is inconsistent.	Verify that the right configuration / calibration for the images is loaded.
CS3D_NO_IMAGE_SET	-72	The source image pointer is NULL or not set.	The source pointer has to be set before calling the “setSrcImgLoaded” function.
CS3D_INVALID_ROI	-71	The ROI is not within the image borders. It has a size of zero or a negative size	Double check the set ROI.
CS3D_CALC_IMAGE_CANCELED	-70	If “cancel” is called while waiting for an image the “getNextImgBlocking” returns that error.	This error appears if you cancel the waiting for an image.
CS3D_CUDA_DRIVER_OUTDATED	-69	The installed GPU drivers are too old.	Please update the drivers of you NVidia-GPU.
CS3D_INVALID_IMAGENUMBER	-68	Tried to access an image that is not in a buffer.	It seems that this image is not calculated or already freed.
CS3D_IMAGE_IN_USE	-67	The calculation on that image is still running.	Only access images if they are already calculated.
CS3D_UNSUPPORTED_IMAGE_FORMAT	-66	Given image type/source image info doesn’t match.	Please check this manual for supported image formats.
CS3D_INVALID_PARAMETER	-65	One or more parameter settings are improper.	Please check the documentation of the function and use it correctly.
CS3D_CAM_CONNECTION_ERROR	-64	Cannot connect to camera	Please check if the camera is properly connected, and if connection type and port are set correctly, or try to update camera firmware.
CS3D_CAM_RW_ERROR	-63	Reading/writing data from/to camera error	Please check if the camera is properly connected, and make sure the data to be uploaded does not exceed the camera memory.
CS3D_IMAGE_LOADING_ERROR	-62	Cannot load the image(s)	Please check if the image file path is correct.

CS3D_CALIB_VERIFICATION_FAILED	-61	Calibration verification failed or has low confidence	Please check the image quality (the image should be sharp and rich in texture along whole image width), or if the image is scanned in wrong direction, or try to adjust the height range.
CS3D_CALIB_TARGET_NOT_FOUND	-60	Calibration target is not or only partially detected in the given image(s).	Check if the target is visible in (both) image(s) and if the illumination is neither to bright nor to dark.
CS3D_TARGET_DEFINITION_ERROR	-59	Calibration definition not valid / not found.	Check if right target definition is set.
CS3D_CALIBRATION_TOO_OLD	-58	Calibration version is too old for the current software.	Use a newer calibration version or downgrade to an older CS-3D software version .
CS3D_CAM_RW_PERMISSION_ERROR _	-57	Unable to write/read data to/from the camera.	This error occurs because of the lack of permission to read and write data into camera or because if the data is already present in the camera. Please refer log file for more details.
CS3D_INTERFACE_VERSION_MISMATCH	-56	The version of the interface (CS3D_MAJOR/MINOR_VERSION) does not match what the application requested during creation	Use the header files of correct version provided with the DLL.
CS3D_SYNC_DIFF_ERR_UNRECTIFIABLE	-55	Unable to rectify the synchronization difference between cameras through calculation in API. The synchronisation difference error value is either greater than maximum allowable synchronisation difference error value or the difference is not consistent when checked in multiple lines of source images.	Create new images as these images are corrupt.
CS3D_CALIBRATION_MISMATCH	-54	The calibration file and the input image are not related to each other.	Please use the right calibration file and input image(s).
CS3D_UNRELATED_INPUT_IMAGES	-53	The left and right images (from master and slave cameras of dual camera setup) are not related to each other.	Please use the right pair of images obtained from a dual camera setup.
CS3D_NO_FIRST_LINE_INFOBLOCK	-52	The camera first line info-block, which is required is not available in the image	Please provide input images with first line info block.

CS3D_NO_EACH_LINE_INFO_BLOCK	-51	The camera each line info-block which is required is not available in the image	Please provide input images with each line info block.
CS3D_MASK_NOT_IMPLEMENTED_FOR_MODE	-50	Currently the image mask functionality is not yet available for the Semi Global Matching method	Please use Block matching method, in case the required output data has to be calculated only for masked area.
CS3D_INFOBLOCK_DIFFERENT_POSITION	-49	Camera info blocks in A and B image are in different positions. They both should be either in first pixels or last pixels.	Please provide input image pair that have info-blocks present in similar location in image.
CS3D_INVALID_VERTICAL_MIRROR	-48	One of the source images are mirrored in vertical direction or the variation of line count values in image is not stable.	Please create new input images as these images are corrupt or missing some image lines.
CS3D_MISSING_IMG_LINES	-47	Source image is missing some lines in-between (missing line-count values).	Please create new input images as given input images are missing image lines.

Warning	Code	Description	Possible actions
CS3D_WARN_INVALID_PARAMETER_ADJUSTED	-10001	A parameter was automatically adjusted to its boundaries.	Please check which parameter was out of bounds and correct it
CS3D_WARN_CALIB_NOT_VALID_ANYMORE	-10002	Calibration file may not be valid anymore for the current camera status.	Please do further test to verify the result, or contact support.
CS3D_WARN_ROI_ADJUSTED	-10003	ROI is defined in a region that is not contained in both images of the stereo pair (overlapping area). Only regions contained in both images are effecting the calculation	No action required. If ROI should match the disparity image, adjust ROI to overlapping area
CS3D_WARN_P3D_NOT_VALID	-10004	One or more point cloud calculations return invalid result.	One or more input raw coordinates may be invalid. Please verify the raw coordinates input, or ignore the invalid values.
CS3D_WARN_RESULT_OUT_OF_RANGE	-10005	One or more results are out of specified height range of this sensor	Check if the input values are right. Results can be inexact.

13.2 LabVIEW

Error	Code	Description	Possible action
CS3D_LV_UNDEFINE	-199	Undefined error in	Please feel free to contact

D		wrapper dll	our support.
CS3D_LV_CANT_LOAD_DLL	-198	Can't load CS-3D-API core	(Re-)Install current CS-3D-Software package.
CS3D_LV_CANT_GET_CONFIG	-197	Can't get a config object from CS-3D-API	Verify that config is loaded correctly.
CS3D_LV_CANT_GET_ACTIVE_CONFIG	-196	Can't get an active config object from CS-3D-API	Verify that config is loaded correctly.
CS3D_LV_INVALID_CAMERA_NUMBER	-195	Invalid camera number	Choose a camera number according to your configuration.
CS3D_LV_INVALID_CHANNEL_ORDER	-194	Invalid channel Order	Choose one of the channel orders available.
CS3D_LV_NO_SRCIMGINFO	-193	“srcImgInfo” hasn't been set yet	Use the Vis in the correct order
CS3D_LV_INVALID_OUTPUT_FORMAT	-192	Invalid output format	Choose an available output format.
CS3D_LV_LOAD_IMG_TIMEOUT	-191	Timeout occurred while waiting for to be able to load next images	Increase timeout / limit the height of your images.
CS3D_LV_INVALID_OUTPUT_ARRAY	-190	Invalid output buffer array	Check if you have enough available RAM
CS3D_LV_ALLOC_ERROR	-189	An error occurred while trying to allocate memory	Check if you have enough available RAM
CS3D_LV_NOT_CONFIG_WIDTH	-188	Image width differs from the one in the config	Choose right camera config
CS3D_LV_INVALID_REFERENCE	-187	Invalid reference/pointer to CS-3D-API	Verify that Api-Core is initialized correct.
CS3D_LV_INVALID_CONF_FILE	-186	Invalid configuration file name in initialization	Doublecheck config filename
CS3D_LV_INVALID_CHANNEL_COUNT	-185	Invalid channel count	Choose correct channel count (1 or 3)
CS3D_LV_INVALID_IMAGE_HEIGHT	-184	Invalid image height	Choose correct image height

CS3D_LV_INVALID_API_FILE_PATH	-183	CS-3D-API core not found	(Re-)Install current CS-3D-Software package.

13.3 HALCON

Error	Code	Description	Possible action
Error loading dll	10999	CS-3D core is not found.	(Re-)Install current CS-3D-Software package.
Can’t get config	10998	Config not found	Double check the config filename
Can’t init system Err: -100/../-103	10998	License problem	Check if your license is still valid.

14 Frequently Asked Questions

14.1 *What does Calculation Speed Depend on*

- Calculation Parameters
  - Height range
  - Results calculated, rectified image, height image, point cloud
  - ROI
  - Whether allow some degree of height resolution reduction
- Hardware
  - GPU
    - Number of CUDA cores
    - Clock frequency
  - CPU
    - Number of cores
    - Frequency
  - RAM access speed
  - Bus bandwidth
- Implementation
  - If possible use at least two threads, one for the image acquisition / loading the images to the API and a second one to get the results from the API.
  - Have a look at our example CS3DApiPerformance that will also give you a hint on how fast you can get.
  - If you measure the speed, always take the mean of a whole sequence of calculated images to have a realistic value.

14.2 *How to Increase the Calculation Speed*

- Select the smallest possible height range
- Switch off the calculation of the results that are not needed e.g. “doCalc3DPoints=0” if the point cloud is not used
- Use the ROI function or directly reduce the image height to the avoid calculation of areas that do not contain any information.
- Allow some degree of height resolution reduction by setting parameter heightResolutionReduction in config file (section [DISPARITY]) to 1. This can be an efficient option to increase the calculation speed when the reduced height resolution still fulfills the requirement of your application.

14.3 *How to Use Multiple GPUs*

Adjust the number of GPUs at configuration and select which GPUs to use.

1) Set “numGPUs” to the number of GPUs you want to use
e.g “numGPUs=2” if you want to use two GPUs

2) Set which GPU use as the first, second, … here the enumeration starts with “0”
e.g. “useGPUs[0]=0” and “useGPUs[1]=1” if you want to use the first two GPUs in same instance.

e.g. “useGPUs[0]=1” and “useGPUs[2]=3” if you want to use the second and the fourth GPU in same instance.

14.4 How to Generate a Debug Log

1) Set “debug=9” in configuration you use
2) Install “DebugView” from http://technet.microsoft.com/enus/sysinternals/bb896647.aspx
3) Start DebugView
4) Execute CS-3D-Api / Viewer

14.5 Support Information

In case you want to ask our support for help regarding a specific software issue or if you like to report an error, please provide us with the following information:

- Serial number and camera type / CP-number
- Image height and width of the source images
- Configuration and calibration file
- CS-3D software version
- Debug log output like described in chapter 14.4
- Does the program work with the sample images?
- Hardware-Info: GPU type / driver version / mainboard type / RAM size
- Detailed error description, screenshots
- Source camera images, if possible

15 Additional remarks

The Chromasens 3D-Api and the Chromasens 3D-Viewer use the OpenCV library http://opencv.org for image processing.

The license of the OpenCV library is included in the “{appDir}/Chromasens/3D/docs” folder.

Chromasens GmbH
Max-Stromeyer-Strasse 116
78467 Konstanz
Germany

Phone: +49 7531 876-0
Fax: +49 7531 876-303

www.chromasens.de
support@chromasens.de