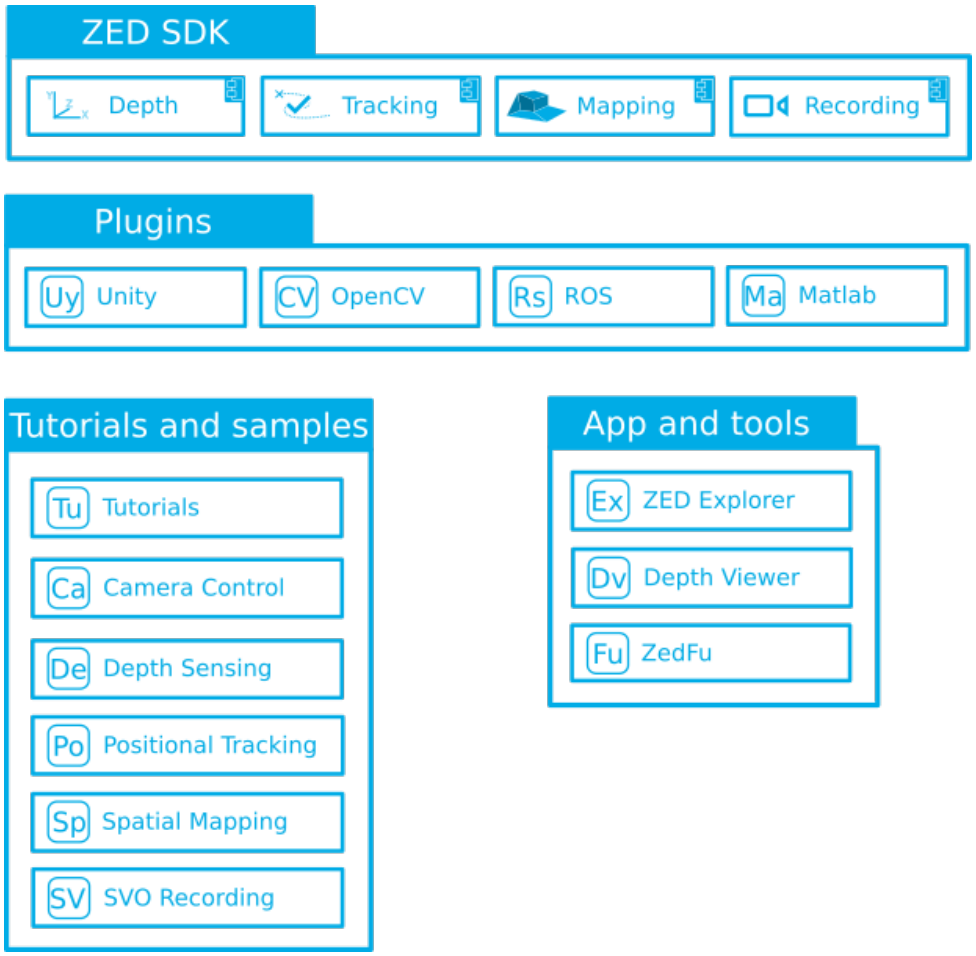
SDK

v2.1.2

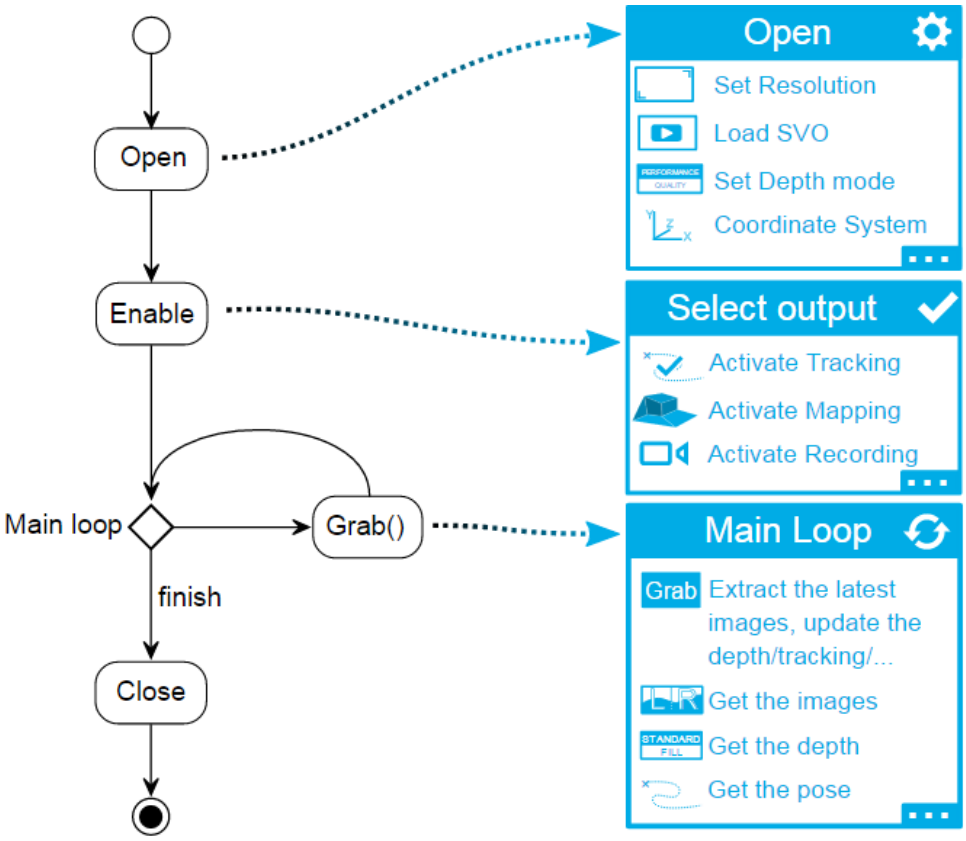
Contents

SDK Introduction



Application life cycle

The ZED SDK provides you several outputs like the depth map, the position/orientation of the camera or the textured point cloud of the scene. The frequency and the accuracy of this data depend on the initial settings that you gave to the SDK. The following scheme resumes its life cycle, it should be a good to keep it aside.



SDK Files

The SDK is installed by default at the following location:

- on **Windows** at C:/Program Files (x86)/ZED SDK
- on **Linux** at /usr/local/zed

It falls into the following directories:

- **app/** contains binary executables of some ZED camera apps. Like ZEDfu
- **bin/** (Windows only) contains the windows DLLs required to use the SDK
- **dependencies/** (Windows only) contains samples and SDK (OpenCV,Eigen...) dependencies
- **doc/** contains documentations and licence
  - **API/** contains the API documentation on Web format
  - **license/** contains the licenses of the SDK and its third-parties'
- **firmware/** contains all firmware version for the ZED camera (*.bin files)

- **include/** contains C/C++ header files
- **lib/** contains the libraries files required to link with the SDK (".lib" on Windows and ".so" on Linux)
- **plugin/** contains the ZED SDK plugins of some major programs
- **sample/** contains sample programs using the SDK
- **settings/** (Linux only) (on Windows at AppData\Roaming\Stereolabs) contains ZED camera parameters
- **tools/** contains binary executables included in the SDK that configure and manage the ZED camera
- **scripts** (Jetson)/ contains performance scripts to be activated for best performances

Release Notes

ZED SDK 2.1.0/1

Major New Features

- **Major update of Depth Sensing API**:
  - New depth stabilization mode is now available. It brings significant improvement to depth sensing accuracy and noise reduction. It is enabled by default and requires the positional tracking (enabled automatically).
  - Depth edges are now significantly sharper in FILL mode.
  - Images/Depth/Measures resolutions can now be adjusted in retrieve∗() functions, speeding up applications that need lower resolution Mat.
  - Normal maps are now available for the left and right eye.
  - Depth map and point cloud are now available for the right eye.

Plugins

- Unity:
  - Added Spatial mapping script and sample.
  - Added Nav Mesh script to enable agent navigation and path finding in spatially mapped areas.
  - Added Light script to enable light interactions between the real and virtual world.
  - Added Vive tracker support.
  - Added Oculus Rift support in Green Screen MR sample.
  - Added a semi-automated calibration script to align the real and virtual world in Green Screen MR sample.

Dependencies

- Removed OpenCV as an SDK dependency.
- OpenCV is still used in Depth sensing and OpenCV samples for image display.

SDK

- Updated Depth API:
  - Added sl::Resolution parameter in retrieve∗() function to allow retrieval of smaller depth resolutions.
  - Updated SENSING_MODE_FILL to improve depth edges and real-time occlusions in mixed-reality applications.
  - Added RuntimeParameters::enable_depth_stabilization to enable depth stabilization. Positional Tracking must be enabled in order to use depth stabilization.
  - Added MEASURE_NORMALS as a new MEASURE to extract the normal map of the scene. Output is an sl::Mat with 4 channels (x,y,z,empty) of 32b float.
  - Added RuntimeParameters::enable_right_measure for all measure types to extract a measure MEASURE_XXX_RIGHT mapped on the right image.

- Updated Spatial Mapping API (see details below):
  - Added vector of sl::Chunk to allow access to smaller parts of a mesh ("chunks").
  - Added Mesh::getVisibilityList() to get the list of chunks visible by the ZED.
  - Added Mesh::getSurroundingList() to get the list of chunks located within a specific distance of the ZED.

Experimental feature

- Spatial Mapping API update:
  - Mesh Chunks have been introduced. Spatial mapping now divides the world in fixed-size blocks called chunks. Instead of storing and accessing a single mesh with a million+ vertices, this allows to store and access multiple local submesh with a much lower number of vertices.
  - Chunks within a certain distance of the camera can be accessed through Visibility and Surrounding lists. See sl::Mesh API documentation for more information.
  - Additional optimization techniques based on chunks will be introduced in later updates.
  - **Known Issue:** The normals can be duplicated with a slight orientation difference on the edge of the chunks on the mesh, this can cause lightning issues

ZED SDK 2.0.1

SDK

- Bug fix on initial_world_transform units
- Bug fix on Mesh::load() function

Tools

- Bug fix on ZEDfu installation for TX1

ZED SDK 2.0.0

Major New Features

- **Major SDK update**:

  - The ZED SDK is now composed of different modules: Video, Depth, Positional Tracking and Spatial Mapping. Each module can be configured through dedicated parameters. Support functions have also been introduced and memory management has been simplified. The new API is much easier to use and integrate.
    * ZED SDK 2.0 introduces breaking changes. To help you switch from 1.2 to 2.0, read the migration guide.

- **New Spatial Mapping module**:
  - New Spatial Mapping module is introduced in 2.0.
  - Spatial Mapping combines stereo depth sensing and motion tracking to capture a 3D map of the environment. Spatial mapping is useful for collision avoidance, motion planning and realistic blending of real and virtual worlds.

- **Major update of Positional Tracking module**:
  - New Spatial Memory mode now enables the ZED to memorize its surroundings. It significantly improves positional tracking by correcting drift and ensures a consistent experience across uses. Spatial Memory is enabled by default and replaces the previous Area Learning mode.

- **New Unity plugin and samples**:
  - Unity package now supports ZED SDK 2.0 and Unity 5.5.
    * Added new VR, MR and Green Screen samples.

SDK

- Updated Video module:
  - Added control of camera White Balance.
  - Reduced video capture latency on Linux: It is now possible to adjust the size of the buffer queue size for video capture using InitParams::reqBuffersCountLinux.
  - Added real-time SVO mode: This mode reads SVO files and skips frames if necessary to simulate the behavior of the camera operating in live mode. Real-time mode is not available for SVO files recorded in RAW compression.
  - Fixed bug with camera control functions that sometimes returned erroneous values.

- Updated Depth module:
  - Improved depth quality for STANDARD mode and reduced aliasing.

- Updated Positional Tracking module:
  - Introduced Spatial Memory mode that significantly reduces drift over time.
  - Introduced new Pose, Translation, Orientation, Rotation and Transform classes that provide better and finer control over the API.
  - Refactored tracking parameters into a TrackingParameters class. Use default parameters for optimal results.

- Added Spatial Mapping module:
  - Introduced new sl::Mesh class to store polygonal meshes.
  - enableSpatialMapping() starts Spatial Mapping.
  - requestMeshAsync(), getMeshRequestStatusAsync(), retrieveMeshAsync(sl::Mesh) request and retrieve the mesh that is being created. These functions are asynchronous and designed for real-time mesh extraction.
  - extractWholeMesh(sl::Mesh) extracts the current mesh synchronously. This function is designed to extract the whole mesh at the end of a mapping session.
  - disableSpatialMapping() stops Spatial Mapping.

- Added new Mat class:
  - The sl::Mat class simplifies CPU and GPU memory management.
  - Memory can be owned or shared (reference) with another Mat.
  - Reallocation is handled by the SDK. It is not necessary to pre-allocate memory.
  - Transfer between CPU and GPU memory is simplified using Mat::updateCPUfromGPU() and Mat::updateGPUfromCPU().
  - Direct conversion to OpenCV cv::Mat is now available using Mat::toCVMat().
  - Direct data access is available using Mat::getPtr<T>()

- General SDK changes:
  - Added default Camera constructor and open()/close() functions
  - Unified namespace: all the class and function are now into the single unified namespace "sl".
  - Changed filename input type from std::string to sl::String, based on const char* to allow Debug build under Windows.
  - Renamed coordinate systems and added new COORDINATE_SYSTEM "LEFT_HANDED_Z_↩ UP".
  - Fixed bug with invalid depth values when out of range.
  - Minor bug fixes and general improvements.

Tools

- ZED Explorer: Updated GUI.
- ZED Depth Viewer: Updated GUI.
- ZEDfu: Updated tool to use the new Spatial Mapping API.
- ZED SVO Editor: Added compression options.
- ZED Calibration: Fixed a crash when adding or removing the ZED while the tool is running.
- Updated Diagnostic Tool.

Plugins

- Unity:
  - Refactored plugin to ensure compatibility with ZED SDK 2.0 and Unity 5.5.
  - Improved overall stability and performance.
  - Added Positional Tracking for VR sample.
  - Added Green Screen VR Capture sample.
  - Added Simple Mixed Reality sample.
- ROS:
  - Updated wrapper to ensure compatibility with ZED SDK 2.0.
- Matlab:
  - Updated interface to ensure compatibility with ZED SDK 2.0.

Samples and Tutorials

- Introduced new samples for each major API module:
  - Camera Control: Shows how to open the ZED, control camera settings and display images.
  - Depth Sensing: Shows how to get a depth map from the ZED and display a 3D point cloud using OpenGL.
  - Motion Tracking: Shows how to use the ZED as a positional tracker and display the ZED in 3D space using OpenGL.
  - Spatial Mapping: Shows how to map a scene using the Spatial Mapping API and display the projection of the mesh on the image in real-time using OpenGL.

- Added new 'Interface' samples:
  - OpenCV: Shows how to use the ZED with OpenCV.
  - PCL: Shows how to use the ZED with PCL.

- Added tutorials showing how to use the different modules of the SDK: Video, Depth, Tracking, Mapping.

Dependencies

- Removed Eigen.
- Removed Boost from all samples.

Platform Support

- New Jetson TX2 is now supported.

ZED SDK 1.2.0

New Features

- **Major update of Unity plugin**:
  - Unity plugin now includes ZED images, depth and tracking, along with new simplified C# interface.

- **Major update of ROS wrapper**:
  - ROS wrapper has been significantly improved. New features have been introduced and documentation has been updated.

- **New recording modes**:
  - A new lossy compression mode is introduced in this release. It improves the compression ratio by 5 with minimal loss in image quality and is faster than previous compression modes.

- **Frame drop detection**:
  - Dropped frames can now be detected by calling getFrameDroppedCount().

SDK

- Improved SVO recording:
  - Added lossy compression mode.
  - Improved lossless compression and limited frame drops during recording.
  - Added recording_state structure for record() function output. The structure contains status, current and average compression time/ratio.

- Bug fixes:
  - Fixed bug when destroying Camera object that could lead to a crash.
  - Fixed bug (Jetson only) when calling retrieveMeasure_cpu() and writePointCloudAs() functions that stored a part of the image.
  - Fixed bug when using setCameraSettingsValue() in SVO mode.

Tools and Samples

- ZED Explorer: Added compression mode selection for recording.
- ZED Explorer: Fixed minor bugs with recording in command line.
- ZED Depth Viewer: Minor bug fixes.
- Code refactoring and documentation update of most samples on GitHub.

App

- ZEDfu: Fixed bug when RAM memory limit is exceeded during mesh post-processing. Now $*$.ply files are saved even if post-processing doesn't run.

Plugins

- Unity: New package which makes ZED stereo images, depth and tracking available in Unity.
- Unity: All major C++ ZED SDK functions are now accessible in Unity through C# interface.
- ROS wrapper: Improved performance and fixed minor bugs.
  - Converted wrapper to nodelet to improve performance. Thanks to GitHub contributor .
  - Added support for using multiple ZED and GPU.
  - Added URDF model.
  - Updated topics names and added additional launch file examples.

Compatibility notes:

- Jetson TK1: This is the last update for Jetson TK1. Next releases will support TX1 only.

ZED SDK 1.1.1

Platforms

- Added Ubuntu 16.04 compatibility .
- Added NVIDIA Pascal GPUs support and CUDA 8 compatibility on Windows and Linux.
- Added JetPack 2.3 compatibility on Jetson TX1.

SDK

- grab() function now returns an appropriate error code ERRCODE when issues occur.
- grab() function now accepts either a structure as input or a list of variables.
- Minor bug fixes.

Tools

- Fixed bug in ZED Calibration tool that could lead to app crash.

App

- ZEDfu: Improved memory management and framerate.
- ZEDfu: Fixed bug when Live input is started and self-calibration fails.

ZED SDK 1.1.0

New Features

- **New recording pipeline**:
  - SVO or AVI recording can now be started whith enableRecording() and stopped with stop↩ Recording() without deleting the ZED camera object. You can also record videos whith depth and tracking enabled. Use the record() function after a grab() call to save incoming images in your video file.

- **New grayscale and raw image output**:
  - Grayscale images and raw (unrectified) images can now be retrieved using retrieveImage∗ functions.

SDK

- New positional tracking matrix output:
  - Besides the 4x4 Eigen matrix that contains the position and orientation of the camera, a new structure "MotionPoseData" that contains the rotation, translation and orientation (quaternion) of the camera can be retrieved. The quaternion can be used more easily in OpenGL and Unity applications.

- New XYZ+Color output:
  - Added XYZBGRA, XYZARGB, XYZABGR support in retrieveMeasure∗() function. This improves performance with ROS and other third-party libraries.

- Bug fixes:
  - Fixed PNG error when SVO end is reached.
  - Fixed some Valgrind errors when using ZED SDK.
  - Fixed timestamp output on Windows (now in nanoseconds).

Tools and Samples

- New diagnostic tool:
  - ZED Diagnostic, located in the Tools folder, analyzes your hardware configuration and detects potential issues that may affect the performance of your system.

- Minor fixes:
  - ZED Depth Viewer window can now be resized.
  - ZED Depth Viewer and ZED Explorer now support ZED disconnections and reconnections.
  - Added frame drop and effective framerate indicators in ZED Explorer for Live, Recording and SVO Playback mode.

Compatibility notes:

- Jetson TX1: Current version is for JetPack 2.1 in 32bits mode. New versions will only be released for JetPack 2.2 and above in 64bits mode.
- Windows: Current version supports CUDA 7.5. Next updates will support CUDA 8.0 only.
- Ubuntu: Current version supports Ubuntu 14.04 and CUDA 7.5. Next updates will support Ubuntu 16.04 with CUDA 8.0.

ZED SDK 1.0.0

Major New Features

- **Positional tracking API**:
  - New tracking API is introduced in 1.0
  - It calculates position and orientation of the camera for every frame using a new improved stereo SLAM technology. See Tracking samples to learn how to use the new API.

- **New ZEDfu large-scale 3D reconstruction application**:
  - ZEDfu demonstrates how to combine stereo depth sensing and camera tracking to capture realistic environmental assets for VR, films and games.
  - Now available as a standalone downloadable application for Windows platform. Also available in the 'App' folder of the ZED SDK.

- **New high-speed Wide VGA mode**:
  - Updated ZED firmware (1142) introduces Wide VGA @100 FPS, replacing previous VGA mode.
  - WVGA resolution is increased to 672∗376 per image with improved image quality and a much wider field of view.
  - Firmware update is done automatically during installation of the ZED SDK (Win, Linux, Jetson).

- **SVO compression**:
  - New SVO lossless compression now saves 66% in file size and processing time.

- **Unity support**:
  - Positional tracking API is compatible with Unity.
  - Unity plugin for ZED is available as a downloadable package on our Developer page.

SDK

- Added positional tracking API:
  - New getPosition() function provides position and orientation of the camera for every frame. Use enableTracking() during initialization to use the Tracking API.
  - enableAreaLearning parameter can be set to activate camera relocalization and loop closure detection during tracking.

- Updated depth sensing mode with up to 2x faster computation:
  - Updated MODE::PERFORMANCE to new faster than real-time stereo matching algorithm.
  - Added MODE::MEDIUM with depth estimation quality and speed similar to the previous M↩ ODE:PERFORMANCE.

- Added manual control for camera exposure time:
  - Exposure time can now be configured manually by the user with setCameraSettingsValue() function.

- Replaced parameters for Init() function with new "InitParams" structure:
  - UNIT: Desired metric units can be specified in “InitParams” structure. All the provided data will be scaled according to the selected units: baseline, depth, XYZ, depthclamp, closestDephValue, tracking information... Therefore, "setBaselineRatio()" has been removed.
  - COORDINATE_SYSTEM: Coordinate systems such as left or right-handed coordinates can now be selected in “InitParams” structure. The oriented data such as (XYZ) point cloud and tracking information will be expressed in the specified coordinate system.
  - minimumDistance: the minimumDistance for depth computation can now be configured manually and set above 50cm. On certain configurations, this can dramatically improve frame-rate.

- Invalid MEASURE values are now separated into three distinct values:
  - OCCLUSION_VALUE: NaN (not a number), for occlusions.
  - TOO_FAR: Inf (infinity), for values above DepthClampValue.
  - TOO_CLOSE: -Inf, for values below ClosestDepthValue.
  - isValidMeasure(): New macro function that returns false if the given value is one of the above, else returns true.

- Added function to calculate ZED camera field of view:
  - Left and right eye field of view can now be retrieved using getParameters() : LeftCam/Right↩ Cam.hFOV/dFOV/vFOV. See StereoParameters structure for more details.

- SVO files now record the camera framerate as well as timestamp during recording:
  - Use getCameraTimestamp() function to extract the timestamp from the current SVO image. This can be used to detect frame drops during recording.

Tools and Samples

- New unified GUI for the tools
- Added two samples using the Positional Tracking API: cpu/Simple Tracking and cpu/Tracking Viewer.
- ‘Grabbing thread’ sample has been renamed to ‘Optimized Grab’.
- Updated all samples with ZED SDK 1.0 compatibility

API Changes

- Camera::init(...): Replaced arguments with InitParams structure. See SDK Changes.
- SENSING_MODE: RAW and FULL modes have been renamed to STANDARD and FILL. This better reflects the use of the different depth sensing modes. Old names have been removed.

Dependencies

- Updated OpenCV, from 2.4.9 to 3.1 on Window and Linux.
- Added Eigen 3.

ZED SDK 0.9.4 Beta

Platforms

- Linux: Added compatibility for Linux Kernel 4.X. Previous ZED SDK version were incompatible with new Linux kernel, producing a "ZED_NOT_AVAILABLE" message and segfault.

SDK

- Fixed a bug in color correction.

Tools

- Introducing a new tool called "ZED Calibration" that can be used to easily recalibrate your ZED. Must be used in Live mode. Please follow the tutorial inside the tool to learn how to use this new tool.
- Added automated check of ZED SDK version in ZED Explorer. You can also use ZED Explorer with -v command line option to check if your SDK is up to date.
- Fixed bug in ZED Depth Viewer that could lead to degraded performance on specific CPU/GPU configurations on Windows.

ZED SDK 0.9.3 Beta

Platforms

- Compatible with CUDA 7.5 for Windows and Linux.

SDK

- Improved grab() time.
- Reduce Latency when using cpu functions for Jetson.
- Changed Selfcalibration to a non-blocking function. Self-Calibration is done in background and the result can be checked with the status function.
- Improved Init() time

Tools

- Merged ZED Explorer and ZED Settings App into one single tool.
- Added command line options for ZED Explorer for check compatibility and recording

Known Issues

- On Ubuntu 14.04, the newer kernel 4.X is NOT supported.

ZED SDK 0.9.2b Beta

SDK

Bug fixes. Patches version.

ZED SDK 0.9.2 Beta

Platforms

- Added Window 10 compatibility.

SDK

- Improved Left/Right image quality with retrieveImage*().
- Improved grab time on embedded platform (Jetson TK1).
- Added possibility to disable self-calibration during init().
- Added function that returns results of self-calibration.
- More calibration parameters available.

Tools

- Bug fixes in Depth Viewer tool.
- Added more calibration parameters in ZED Settings App

ZED SDK 0.9.1 Beta

SDK

- Bug fixes for Auto-calibration that made samples crash on Windows 8
- Bug fixes for ply write function that made incoherent color.
- Add more support for slMat2cvMat function.

Tools

- Improved global framerate of Depth Viewer tool.

ZED SDK 0.9.0 Beta

SDK

- Added triangulation function to extract XYZ or XYZRGBA point cloud.
- Added Save functions to save point cloud in multiple formats and depth image in multiple format.
- Added Camera and Current Timestamp extraction function to help synchronization with other devices.
- Added FPS extraction function.
- Bug fixes when using svo files.
- Improved AutoCalibration.

Samples

- Complete re-factory of samples.
- Added Cuda sample for background substraction and image disparity for right image.
- Added ROS Sample to demonstrate how to interact with ROS.
- Added Recording sample to demonstrate how to use recording functions.
- Added SVO Playback sample to demonstrate how to simply use the svo file.
- Simplify existing samples.

Tools

- Added ZED Depth Viewer sample to have a plug and play tool to demonstrate all the functions of the ZED SDK.
- Added Cmd Line mode for ZED Explorer to record svo without Graphics.

ZED SDK 0.8.2 Beta

SDK

- Added flip mode (to work with ZED Camera vertically flipped)
- Added multiple input possibility under Linux system.

Samples

- Added Multiple input sample.

ZED SDK 0.8.1 Beta

SDK

- Major improvement of Disparity estimation in untextured areas and repetitive patterns
- Reduced grab() time by 10%
- Reduced retrieveImage() time to $<$0.5ms when grab is done with disparity enabled
- Added new function to adjust ZED camera parameters (set / getCameraSettingsValue), including exposure, white balance, brightness, contrast and hue
- Added option to select a different framerate other than the default ones available in the ZED Camera constructor. For example, it is now possible to select VGA mode @ 30 fps
- Added getSVOPosition() to retrieve current SVO position
- Bug fix in 'Live mode' on Windows x64
- Added compatibility with NVIDIA GeForce 9 series of graphics cards

Tools

- Minor bug fix in ZED Explorer on Linux

ZED SDK 0.8.0 Beta

Platforms

- Added Linux (Ubuntu 14.04 LTS only) compatibility.

Tools

- Added SVOEditor: command line tool to cut and concatenate SVO files. See -help for more details
- Bug fix in ZED Settings App with "-sn" options
- Minor bug fix in ZED Explorer

SDK

- Improved Disparity estimation
- Bug fix in Disparity/Depth normalization
- Bug fix in NONE grab mode (MODE::NONE)
- Merged MODE::MEDIUM et MODE::QUALITY into a single QUALITY MODE
- Add specific functions for SVO (set position and get number of frames)
- Changed name of "DispReliability" to "ConfidenceThreshold"
- Conformed set/get functions

Known Issues

- Tools or Samples may not launch when run twice on Win8. Reconnecting the ZED will solve this issue.

ZED SDK 0.7.1a Alpha

Notes

- Initial release.

Known Issues

- Specific samples can crash at launch on Win 8. Relaunching the sample solves the problem.

Deprecated List

Member Orientation::getRotation () const

Member Orientation::setRotation (const Rotation &rotation)

Member Pose::getRotation ()

Member RuntimeParameters::enable_point_cloud
: this parameter is deprecated as of ZED SDK 2.1. Point cloud is now enabled once depth is.

Member Transform::getRotation () const

Member Transform::setRotation (const Rotation &rotation)

Module Index

Modules

Here is a list of all modules:

Namespace Index

sl Namespace Reference

Classes

- struct CalibrationParameters
  Intrinsic parameters of each cameras and extrinsic (translation and rotation).
- class Camera
  The main class to use the ZED camera.
- struct CameraInformation
  Camera specific parameters.
- struct CameraParameters
  Intrinsic parameters of a camera.
- class Chunk
  Represents a sub mesh, it contains local vertices and triangles.
- class InitParameters
  Parameters that will be fixed for the whole execution life time of the sl::Camera.

- class Mat

  The Mat class can handle multiple matrix format from 1 to 4 channels, with different value types (float or uchar), and can be stored CPU and/or GPU side.
- class Matrix3f

  Represents a generic three-dimensional matrix.
- class Matrix4f

  Represents a generic fourth-dimensional matrix.
- class Mesh

  A mesh contains the geometric (and optionally texturing) data of the scene computed by the spatial mapping.
- class MeshFilterParameters

  Parameters for the optional filtering step of a sl::Mesh.
  A default constructor is enabled and set to its default parameters.
- class Orientation

  Designed to contain orientation (quaternion) data of the positional tracking.
- class Pose

  Contains the positional tracking data which gives the position and orientation of the ZED in 3D space.
- struct RecordingState

  Recording structure that contains information about SVO.
- struct Resolution

  Width and height of an array.
- class Rotation

  Designed to contain rotation data of the positional tracking. It inherits from the generic sl::Matrix3f.
- class RuntimeParameters

  Parameters that defines the behavior of the sl::Camera::grab().
- class SpatialMappingParameters

  Sets the spatial mapping parameters.
- class String

  Defines a string.
- class Texture

  Contains information about texture image associated to a sl::Mesh.
- class TrackingParameters

  Parameters for ZED tracking initialization.
- class Transform

  Designed to contain translation and rotation data of the positional tracking.
- class Translation

  Designed to contain translation data of the positional tracking.
- class Vector2

  Represents a two dimensions vector for both CPU and GPU.
- class Vector3

  Represents a three dimensions vector for both CPU and GPU.
- class Vector4

  Represents a four dimensions vector for both CPU and GPU.

Typedefs

Types definition

- typedef float float1
- typedef Vector2< float > float2
- typedef Vector3< float > float3
- typedef Vector4< float > float4
- typedef unsigned char uchar1

- typedef Vector2< unsigned char > uchar2
- typedef Vector3< unsigned char > uchar3
- typedef Vector4< unsigned char > uchar4
- typedef double double1
- typedef Vector2< double > double2
- typedef Vector3< double > double3
- typedef Vector4< double > double4
- typedef unsigned int uint1
- typedef Vector2< unsigned int > uint2
- typedef Vector3< unsigned int > uint3
- typedef Vector4< unsigned int > uint4
- typedef unsigned long long timeStamp

Enumerations

- enum ERROR_CODE {
  SUCCESS, ERROR_CODE_FAILURE, ERROR_CODE_NO_GPU_COMPATIBLE, ERROR_CODE_N↩
  OT_ENOUGH_GPUMEM,
  ERROR_CODE_CAMERA_NOT_DETECTED, ERROR_CODE_INVALID_RESOLUTION, ERROR_C↩
  ODE_LOW_USB_BANDWIDTH, ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE,
  ERROR_CODE_INVALID_SVO_FILE, ERROR_CODE_SVO_RECORDING_ERROR, ERROR_CODE↩
  _INVALID_COORDINATE_SYSTEM, ERROR_CODE_INVALID_FIRMWARE,
  ERROR_CODE_INVALID_FUNCTION_PARAMETERS, ERROR_CODE_NOT_A_NEW_FRAME, ER↩
  ROR_CODE_CUDA_ERROR, ERROR_CODE_CAMERA_NOT_INITIALIZED,
  ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE, ERROR_CODE_INVALID_FUNCTION_CALL,
  ERROR_CODE_CORRUPTED_SDK_INSTALLATION, ERROR_CODE_LAST }

  List error codes in the ZED SDK.
- enum MESH_FILE_FORMAT { MESH_FILE_PLY, MESH_FILE_PLY_BIN, MESH_FILE_OBJ, MESH_F↩
  ILE_LAST }

  List available mesh file formats.
- enum MESH_TEXTURE_FORMAT { MESH_TEXTURE_RGB, MESH_TEXTURE_RGBA, MESH_TEX↩
  TURE_LAST }

  List available mesh texture formats.
- enum MEM { MEM_CPU = 1, MEM_GPU = 2 }

  List available memory type.
- enum COPY_TYPE { COPY_TYPE_CPU_CPU, COPY_TYPE_CPU_GPU, COPY_TYPE_GPU_GPU, C↩
  OPY_TYPE_GPU_CPU }

  List available copy operation on Mat.
- enum MAT_TYPE {
  MAT_TYPE_32F_C1, MAT_TYPE_32F_C2, MAT_TYPE_32F_C3, MAT_TYPE_32F_C4,
  MAT_TYPE_8U_C1, MAT_TYPE_8U_C2, MAT_TYPE_8U_C3, MAT_TYPE_8U_C4 }

  List available Mat formats.
- enum RESOLUTION {
  RESOLUTION_HD2K, RESOLUTION_HD1080, RESOLUTION_HD720, RESOLUTION_VGA,
  RESOLUTION_LAST }

  Represents the available resolution defined in sl::cameraResolution.
- enum CAMERA_SETTINGS {
  CAMERA_SETTINGS_BRIGHTNESS, CAMERA_SETTINGS_CONTRAST, CAMERA_SETTINGS_H↩
  UE, CAMERA_SETTINGS_SATURATION,
  CAMERA_SETTINGS_GAIN, CAMERA_SETTINGS_EXPOSURE, CAMERA_SETTINGS_WHITEBA↩
  LANCE, CAMERA_SETTINGS_AUTO_WHITEBALANCE,
  CAMERA_SETTINGS_LAST }

  List available camera settings for the ZED camera (contrast, hue, saturation, gain...).
- enum SELF_CALIBRATION_STATE {
  SELF_CALIBRATION_STATE_NOT_STARTED, SELF_CALIBRATION_STATE_RUNNING, SELF_C↩
  ALIBRATION_STATE_FAILED, SELF_CALIBRATION_STATE_SUCCESS,
  SELF_CALIBRATION_STATE_LAST }

Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the sl::Camera::open or Reset function.

- enum DEPTH_MODE {
  DEPTH_MODE_NONE, DEPTH_MODE_PERFORMANCE, DEPTH_MODE_MEDIUM, DEPTH_M↩
  ODE_QUALITY,
  DEPTH_MODE_LAST }

  List available depth computation modes.
- enum SENSING_MODE { SENSING_MODE_STANDARD, SENSING_MODE_FILL, SENSING_MOD↩
  E_LAST }

  List available depth sensing modes.
- enum UNIT {
  UNIT_MILLIMETER, UNIT_CENTIMETER, UNIT_METER, UNIT_INCH,
  UNIT_FOOT, UNIT_LAST }

  List available unit for measures.
- enum COORDINATE_SYSTEM {
  COORDINATE_SYSTEM_IMAGE, COORDINATE_SYSTEM_LEFT_HANDED_Y_UP, COORDINAT↩
  E_SYSTEM_RIGHT_HANDED_Y_UP, COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP,
  COORDINATE_SYSTEM_LEFT_HANDED_Z_UP, COORDINATE_SYSTEM_LAST }

  List available coordinates systems for positional tracking and 3D measures.
- enum MEASURE {
  MEASURE_DISPARITY, MEASURE_DEPTH, MEASURE_CONFIDENCE, MEASURE_XYZ,
  MEASURE_XYZRGBA, MEASURE_XYZBGRA, MEASURE_XYZARGB, MEASURE_XYZABGR,
  MEASURE_NORMALS, MEASURE_DISPARITY_RIGHT, MEASURE_DEPTH_RIGHT, MEASURE_X↩
  YZ_RIGHT,
  MEASURE_XYZRGBA_RIGHT, MEASURE_XYZBGRA_RIGHT, MEASURE_XYZARGB_RIGHT, ME↩
  ASURE_XYZABGR_RIGHT,
  MEASURE_NORMALS_RIGHT, MEASURE_LAST }

  List retrievable measures.
- enum VIEW {
  VIEW_LEFT, VIEW_RIGHT, VIEW_LEFT_GRAY, VIEW_RIGHT_GRAY,
  VIEW_LEFT_UNRECTIFIED, VIEW_RIGHT_UNRECTIFIED, VIEW_LEFT_UNRECTIFIED_GRAY, V↩
  IEW_RIGHT_UNRECTIFIED_GRAY,
  VIEW_SIDE_BY_SIDE, VIEW_DEPTH, VIEW_CONFIDENCE, VIEW_NORMALS,
  VIEW_DEPTH_RIGHT, VIEW_NORMALS_RIGHT, VIEW_LAST }

  List available views.
- enum DEPTH_FORMAT { DEPTH_FORMAT_PNG, DEPTH_FORMAT_PFM, DEPTH_FORMAT_P↩
  GM, DEPTH_FORMAT_LAST }

  List available file formats for saving depth maps.
- enum POINT_CLOUD_FORMAT {
  POINT_CLOUD_FORMAT_XYZ_ASCII, POINT_CLOUD_FORMAT_PCD_ASCII, POINT_CLOUD_↩
  FORMAT_PLY_ASCII, POINT_CLOUD_FORMAT_VTK_ASCII,
  POINT_CLOUD_FORMAT_LAST }

  List available file formats for saving point clouds. Stores the spatial coordinates (x,y,z) of each pixel and optionally its RGB color.
- enum TRACKING_STATE {
  TRACKING_STATE_SEARCHING, TRACKING_STATE_OK, TRACKING_STATE_OFF, TRACKING↩
  _STATE_FPS_TOO_LOW,
  TRACKING_STATE_LAST }

  List the different states of positional tracking.
- enum AREA_EXPORT_STATE {
  AREA_EXPORT_STATE_SUCCESS, AREA_EXPORT_STATE_RUNNING, AREA_EXPORT_STATE_↩
  NOT_STARTED, AREA_EXPORT_STATE_FILE_EMPTY,
  AREA_EXPORT_STATE_FILE_ERROR, AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED,
  AREA_EXPORT_STATE_LAST }

  List the different states of spatial memory area export.

- enum REFERENCE_FRAME { REFERENCE_FRAME_WORLD, REFERENCE_FRAME_CAMERA, RE↩ FERENCE_FRAME_LAST }

  Define which type of position matrix is used to store camera path and pose.
- enum SPATIAL_MAPPING_STATE { SPATIAL_MAPPING_STATE_INITIALIZING, SPATIAL_MAPPING_STATE_OK, SPATIAL_MAPPIN↩ G_STATE_NOT_ENOUGH_MEMORY, SPATIAL_MAPPING_STATE_NOT_ENABLED, SPATIAL_MAPPING_STATE_FPS_TOO_LOW, SPATIAL_MAPPING_STATE_LAST }

  Gives the spatial mapping state.
- enum SVO_COMPRESSION_MODE { SVO_COMPRESSION_MODE_RAW, SVO_COMPRESSION↩ _MODE_LOSSLESS, SVO_COMPRESSION_MODE_LOSSY, SVO_COMPRESSION_MODE_LAST }

  List available compression modes for SVO recording.

Functions

- void sleep_ms (int time)

  Tells the program to wait for x ms.
- SL_SDK_EXPORT bool saveDepthAs (sl::Camera &zed, sl::DEPTH_FORMAT format, sl::String name, float factor=1.)

  Writes the current depth map into a file.
- SL_SDK_EXPORT bool saveDepthAs (sl::Mat &depth, sl::DEPTH_FORMAT format, sl::String name, float factor=1.)

  Writes the current depth map into a file.
- SL_SDK_EXPORT bool savePointCloudAs (sl::Camera &zed, sl::POINT_CLOUD_FORMAT format, sl::String name, bool with_color=false, bool keep_occluded_point=false)

  Writes the current point cloud into a file.
- SL_SDK_EXPORT bool savePointCloudAs (sl::Mat &cloud, sl::POINT_CLOUD_FORMAT format, sl↩ ::String name, bool with_color=false, bool keep_occluded_point=false)

  Writes the current point cloud into a file.
- static timeStamp getCurrentTimeStamp ()

  Returns the current timestamp at the time the function is called. Can be compared to sl::Camera::getCamera↩ Timestamp for synchronization.

Enumeration conversion

- static std::string errorCode2str (ERROR_CODE err)

  Converts the given ERROR_CODE into a string.
- static std::string resolution2str (RESOLUTION res)

  Converts the given RESOLUTION into a string.
- static std::string statusCode2str (SELF_CALIBRATION_STATE state)

  Converts the given SELF_CALIBRATION_STATE into a string.
- static DEPTH_MODE str2mode (std::string mode)

  Converts the given string into a DEPTH_MODE.
- static std::string depthMode2str (DEPTH_MODE mode)

  Converts the given DEPTH_MODE into a string.
- static std::string sensingMode2str (SENSING_MODE mode)

  Converts the given SENSING_MODE into a string.
- static std::string unit2str (UNIT unit)

  Converts the given UNIT into a string.
- static UNIT str2unit (std::string unit)

  Converts the given string into a UNIT.
- static std::string trackingState2str (TRACKING_STATE state)

  Converts the given TRACKING_STATE into a string.
- static std::string spatialMappingState2str (SPATIAL_MAPPING_STATE state)

  Converts the given SPATIAL_MAPPING_STATE into a string.

Variables

Unavailable Values

- static const float TOO_FAR = INFINITY
- static const float TOO_CLOSE = -INFINITY
- static const float OCCLUSION_VALUE = NAN

ZED Camera Resolution

- static const std::vector< std::pair< int, int > > cameraResolution

Typedef Documentation

float1

```
typedef float float1
```

float2

```
typedef Vector2<float> float2
```

float3

```
typedef Vector3<float> float3
```

float4

```
typedef Vector4<float> float4
```

uchar1

```
typedef unsigned char uchar1
```

uchar2

```
typedef Vector2<unsigned char> uchar2
```

uchar3

```
typedef Vector3<unsigned char> uchar3
```

uchar4

```
typedef Vector4<unsigned char> uchar4
```

double1

```
typedef double double1
```

double2

```
typedef Vector2<double> double2
```

double3

```
typedef Vector3<double> double3
```

double4

```
typedef Vector4<double> double4
```

uint1

```
typedef unsigned int uint1
```

uint2

```
typedef Vector2<unsigned int> uint2
```

uint3

```
typedef Vector3<unsigned int> uint3
```

uint4

```
typedef Vector4<unsigned int> uint4
```

timeStamp

```
typedef unsigned long long timeStamp
```

Variable Documentation

TOO_FAR

```
const float TOO_FAR = INFINITY  [static]
```

Defines an unavailable depth value that is above the depth Max value.

TOO_CLOSE

```
const float TOO_CLOSE = -INFINITY  [static]
```

Defines an unavailable depth value that is below the depth Min value.

OCCLUSION_VALUE

```
const float OCCLUSION_VALUE = NAN  [static]
```

Defines an unavailable depth value that is on an occluded image area.

cameraResolution

```
const std::vector<std::pair<int, int> > cameraResolution  [static]
```

Initial value:

```
= {
        std::make_pair(2208, 1242),
        std::make_pair(1920, 1080),
        std::make_pair(1280, 720),
        std::make_pair(672, 376)
    }
```

Available video modes for the ZED camera.

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Classes

Here are the classes, structs, unions and interfaces with brief descriptions:

Module Documentation

Public functions

Functions

- void sleep_ms (int time)

  Tells the program to wait for x ms.
- SL_SDK_EXPORT bool saveDepthAs (sl::Camera &zed, sl::DEPTH_FORMAT format, sl::String name, float factor=1.)

  Writes the current depth map into a file.
- SL_SDK_EXPORT bool saveDepthAs (sl::Mat &depth, sl::DEPTH_FORMAT format, sl::String name, float factor=1.)

  Writes the current depth map into a file.
- SL_SDK_EXPORT bool savePointCloudAs (sl::Camera &zed, sl::POINT_CLOUD_FORMAT format, sl::String name, bool with_color=false, bool keep_occluded_point=false)

  Writes the current point cloud into a file.
- SL_SDK_EXPORT bool savePointCloudAs (sl::Mat &cloud, sl::POINT_CLOUD_FORMAT format, sl↩ ::String name, bool with_color=false, bool keep_occluded_point=false)

  Writes the current point cloud into a file.
- static timeStamp getCurrentTimeStamp ()

  Returns the current timestamp at the time the function is called. Can be compared to sl::Camera::getCamera↩ Timestamp for synchronization.

Enumeration conversion

- static std::string errorCode2str (ERROR_CODE err)
  Converts the given ERROR_CODE into a string.
- static std::string resolution2str (RESOLUTION res)
  Converts the given RESOLUTION into a string.
- static std::string statusCode2str (SELF_CALIBRATION_STATE state)
  Converts the given SELF_CALIBRATION_STATE into a string.
- static DEPTH_MODE str2mode (std::string mode)
  Converts the given string into a DEPTH_MODE.
- static std::string depthMode2str (DEPTH_MODE mode)
  Converts the given DEPTH_MODE into a string.
- static std::string sensingMode2str (SENSING_MODE mode)
  Converts the given SENSING_MODE into a string.
- static std::string unit2str (UNIT unit)
  Converts the given UNIT into a string.
- static UNIT str2unit (std::string unit)
  Converts the given string into a UNIT.
- static std::string trackingState2str (TRACKING_STATE state)
  Converts the given TRACKING_STATE into a string.
- static std::string spatialMappingState2str (SPATIAL_MAPPING_STATE state)
  Converts the given SPATIAL_MAPPING_STATE into a string.

Detailed Description

Function Documentation

errorCode2str()

```
static std::string sl::errorCode2str (
            ERROR_CODE err )  [inline], [static]
```

Converts the given ERROR_CODE into a string.

Parameters

err	: a specific ERROR_CODE

Returns

The corresponding string

sleep_ms()

```
void sl::sleep_ms (
            int time )  [inline]
```

Tells the program to wait for x ms.

Parameters

time	: the number of ms to wait.

saveDepthAs() [1/2]

```
SL_SDK_EXPORT bool sl::saveDepthAs (
            sl::Camera & zed,
            sl::DEPTH_FORMAT format,
            sl::String name,
            float factor = 1.  )
```

Writes the current depth map into a file.

Parameters

zed	: the current camera object.
format	: the depth file format you desired.
name	: the name (path) in which the depth will be saved.
factor	: only for PNG and PGM, apply a gain to the depth value. default : 1. The maximum value is 65536, so you can set the Camera::setDepthClampValue to 20000 and give a factor to 3, Do not forget to scale (by 1./factor) the pixel value to get the real depth. The occlusions are represented by 0.

Returns

False if something wrong happen, else return true.

Referenced by Camera::isOpened().

saveDepthAs() [2/2]

```
SL_SDK_EXPORT bool sl::saveDepthAs (
            sl::Mat & depth,
            sl::DEPTH_FORMAT format,
            sl::String name,
            float factor = 1.  )
```

Writes the current depth map into a file.

Parameters

depth	: the depth map to record (CPU 32F_C1 sl::Mat)
format	: the depth file format you desired.
name	: the name (path) in which the depth will be saved.
factor	: only for PNG and PGM, apply a gain to the depth value. default : 1. The maximum value is 65536, so you can set the Camera::setDepthClampValue to 20000 and give a factor to 3, Do not forget to scale (by 1./factor) the pixel value to get the real depth. The occlusions are represented by 0.

Returns

False if something wrong happen, else return true.

savePointCloudAs() [1/2]

```
SL_SDK_EXPORT bool sl::savePointCloudAs (
            sl::Camera & zed,
            sl::POINT_CLOUD_FORMAT format,
            sl::String name,
```

```
            bool with_color = false,
            bool keep_occluded_point = false )
```

Writes the current point cloud into a file.

Parameters

zed	: the current camera object.
format	: the point cloud file format you desired.
name	: the name (path) in which the point cloud will be saved.
with_color	: indicates if the color must be saved. default : false.
keep_occluded_point	: indicates if the non available data should be saved and set to 0, if set to true this give a Point Cloud with a size = height $*$ width. default : false.

Returns

False if something wrong happen, else return true.

Note

The color is not saved for XYZ and VTK files.

Referenced by Camera::isOpened().

savePointCloudAs() [2/2]

```
SL_SDK_EXPORT bool sl::savePointCloudAs (
            sl::Mat & cloud,
            sl::POINT_CLOUD_FORMAT format,
            sl::String name,
            bool with_color = false,
            bool keep_occluded_point = false )
```

Writes the current point cloud into a file.

Parameters

cloud	: the point cloud to record (CPU 32F_C4 sl::Mat)
format	: the point cloud file format you desired.
name	: the name (path) in which the point cloud will be saved.
with_color	: indicates if the color must be saved. default : false.
keep_occluded_point	: indicates if the non available data should be saved and set to 0, if set to true this give a Point Cloud with a size = height $*$ width. default : false.

Returns

False if something wrong happen, else return true.

Note

The color is not saved for XYZ and VTK files.

getCurrentTimeStamp()

```
static timeStamp sl::getCurrentTimeStamp ( )  [inline], [static]
```

Returns the current timestamp at the time the function is called. Can be compared to sl::Camera::get↩ CameraTimestamp for synchronization.

Use this function to compare the current timestamp and the camera timestamp, since they have the same reference (Computer start time).

Returns

The current timestamp in ns.

resolution2str()

```
static std::string sl::resolution2str (
            RESOLUTION res )  [inline], [static]
```

Converts the given RESOLUTION into a string.

Parameters

res	: a specific RESOLUTION

Returns

The corresponding string

statusCode2str()

```
static std::string sl::statusCode2str (
            SELF_CALIBRATION_STATE state )  [inline], [static]
```

Converts the given SELF_CALIBRATION_STATE into a string.

Parameters

state	: a specific SELF_CALIBRATION_STATE

Returns

The corresponding string

str2mode()

```
static DEPTH_MODE sl::str2mode (
            std::string mode )  [inline], [static]
```

Converts the given string into a DEPTH_MODE.

Parameters

mode	: a specific depth

Returns

The corresponding DEPTH_MODE

depthMode2str()

```
static std::string sl::depthMode2str (
            DEPTH_MODE mode )  [inline], [static]
```

Converts the given DEPTH_MODE into a string.

Parameters

mode	: a specific DEPTH_MODE

Returns

The corresponding string

sensingMode2str()

```
static std::string sl::sensingMode2str (
            SENSING_MODE mode )  [inline], [static]
```

Converts the given SENSING_MODE into a string.

Parameters

mode	: a specific SENSING_MODE

Returns

The corresponding string

unit2str()

```
static std::string sl::unit2str (
            UNIT unit )  [inline], [static]
```

Converts the given UNIT into a string.

Parameters

unit	: a specific UNIT

Returns

The corresponding string

str2unit()

```
static UNIT sl::str2unit (
            std::string unit )  [inline], [static]
```

Converts the given string into a UNIT.

Parameters

unit	: a specific unit string

Returns

The corresponding UNIT

trackingState2str()

```
static std::string sl::trackingState2str (
            TRACKING_STATE state )  [inline], [static]
```

Converts the given TRACKING_STATE into a string.

Parameters

state	: a specific TRACKING_STATE

Returns

The corresponding string

spatialMappingState2str()

```
static std::string sl::spatialMappingState2str (
            SPATIAL_MAPPING_STATE state )  [inline], [static]
```

Converts the given SPATIAL_MAPPING_STATE into a string.

Parameters

state	: a specific SPATIAL_MAPPING_STATE

Returns

The corresponding string

Public enumerations

Enumerations

- enum ERROR_CODE {
  SUCCESS, ERROR_CODE_FAILURE, ERROR_CODE_NO_GPU_COMPATIBLE, ERROR_CODE_N↩
  OT_ENOUGH_GPUMEM,
  ERROR_CODE_CAMERA_NOT_DETECTED, ERROR_CODE_INVALID_RESOLUTION, ERROR_C↩
  ODE_LOW_USB_BANDWIDTH, ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE,
  ERROR_CODE_INVALID_SVO_FILE, ERROR_CODE_SVO_RECORDING_ERROR, ERROR_CODE↩
  _INVALID_COORDINATE_SYSTEM, ERROR_CODE_INVALID_FIRMWARE,
  ERROR_CODE_INVALID_FUNCTION_PARAMETERS, ERROR_CODE_NOT_A_NEW_FRAME, ER↩
  ROR_CODE_CUDA_ERROR, ERROR_CODE_CAMERA_NOT_INITIALIZED,
  ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE, ERROR_CODE_INVALID_FUNCTION_CALL,
  ERROR_CODE_CORRUPTED_SDK_INSTALLATION, ERROR_CODE_LAST }

  List error codes in the ZED SDK.
- enum MESH_FILE_FORMAT { MESH_FILE_PLY, MESH_FILE_PLY_BIN, MESH_FILE_OBJ, MESH_F↩
  ILE_LAST }

  List available mesh file formats.
- enum MESH_TEXTURE_FORMAT { MESH_TEXTURE_RGB, MESH_TEXTURE_RGBA, MESH_TEX↩
  TURE_LAST }

  List available mesh texture formats.
- enum FILTER { FILTER_LOW, FILTER_MEDIUM, FILTER_HIGH }

  List available mesh filtering intensity.
- enum RESOLUTION { RESOLUTION_HIGH, RESOLUTION_MEDIUM, RESOLUTION_LOW }

  List the spatial mapping resolution presets.
- enum RANGE { RANGE_NEAR, RANGE_MEDIUM, RANGE_FAR }

  List the spatial mapping depth range presets.
- enum MEM { MEM_CPU = 1, MEM_GPU = 2 }

  List available memory type.
- enum COPY_TYPE { COPY_TYPE_CPU_CPU, COPY_TYPE_CPU_GPU, COPY_TYPE_GPU_GPU, C↩
  OPY_TYPE_GPU_CPU }

  List available copy operation on Mat.
- enum MAT_TYPE {
  MAT_TYPE_32F_C1, MAT_TYPE_32F_C2, MAT_TYPE_32F_C3, MAT_TYPE_32F_C4,
  MAT_TYPE_8U_C1, MAT_TYPE_8U_C2, MAT_TYPE_8U_C3, MAT_TYPE_8U_C4 }

  List available Mat formats.
- enum RESOLUTION {
  RESOLUTION_HD2K, RESOLUTION_HD1080, RESOLUTION_HD720, RESOLUTION_VGA,
  RESOLUTION_LAST }

  Represents the available resolution defined in sl::cameraResolution.
- enum CAMERA_SETTINGS {
  CAMERA_SETTINGS_BRIGHTNESS, CAMERA_SETTINGS_CONTRAST, CAMERA_SETTINGS_H↩
  UE, CAMERA_SETTINGS_SATURATION,
  CAMERA_SETTINGS_GAIN, CAMERA_SETTINGS_EXPOSURE, CAMERA_SETTINGS_WHITEBA↩
  LANCE, CAMERA_SETTINGS_AUTO_WHITEBALANCE,
  CAMERA_SETTINGS_LAST }

  List available camera settings for the ZED camera (contrast, hue, saturation, gain...).
- enum SELF_CALIBRATION_STATE {
  SELF_CALIBRATION_STATE_NOT_STARTED, SELF_CALIBRATION_STATE_RUNNING, SELF_C↩
  ALIBRATION_STATE_FAILED, SELF_CALIBRATION_STATE_SUCCESS,
  SELF_CALIBRATION_STATE_LAST }

  Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the sl::Camera::open or Reset function.

- enum DEPTH_MODE {
  DEPTH_MODE_NONE, DEPTH_MODE_PERFORMANCE, DEPTH_MODE_MEDIUM, DEPTH_M↩
  ODE_QUALITY,
  DEPTH_MODE_LAST }

  List available depth computation modes.
- enum SENSING_MODE { SENSING_MODE_STANDARD, SENSING_MODE_FILL, SENSING_MOD↩
  E_LAST }

  List available depth sensing modes.
- enum UNIT {
  UNIT_MILLIMETER, UNIT_CENTIMETER, UNIT_METER, UNIT_INCH,
  UNIT_FOOT, UNIT_LAST }

  List available unit for measures.
- enum COORDINATE_SYSTEM {
  COORDINATE_SYSTEM_IMAGE, COORDINATE_SYSTEM_LEFT_HANDED_Y_UP, COORDINAT↩
  E_SYSTEM_RIGHT_HANDED_Y_UP, COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP,
  COORDINATE_SYSTEM_LEFT_HANDED_Z_UP, COORDINATE_SYSTEM_LAST }

  List available coordinates systems for positional tracking and 3D measures.
- enum MEASURE {
  MEASURE_DISPARITY, MEASURE_DEPTH, MEASURE_CONFIDENCE, MEASURE_XYZ,
  MEASURE_XYZRGBA, MEASURE_XYZBGRA, MEASURE_XYZARGB, MEASURE_XYZABGR,
  MEASURE_NORMALS, MEASURE_DISPARITY_RIGHT, MEASURE_DEPTH_RIGHT, MEASURE_X↩
  YZ_RIGHT,
  MEASURE_XYZRGBA_RIGHT, MEASURE_XYZBGRA_RIGHT, MEASURE_XYZARGB_RIGHT, ME↩
  ASURE_XYZABGR_RIGHT,
  MEASURE_NORMALS_RIGHT, MEASURE_LAST }

  List retrievable measures.
- enum VIEW {
  VIEW_LEFT, VIEW_RIGHT, VIEW_LEFT_GRAY, VIEW_RIGHT_GRAY,
  VIEW_LEFT_UNRECTIFIED, VIEW_RIGHT_UNRECTIFIED, VIEW_LEFT_UNRECTIFIED_GRAY, V↩
  IEW_RIGHT_UNRECTIFIED_GRAY,
  VIEW_SIDE_BY_SIDE, VIEW_DEPTH, VIEW_CONFIDENCE, VIEW_NORMALS,
  VIEW_DEPTH_RIGHT, VIEW_NORMALS_RIGHT, VIEW_LAST }

  List available views.
- enum DEPTH_FORMAT { DEPTH_FORMAT_PNG, DEPTH_FORMAT_PFM, DEPTH_FORMAT_P↩
  GM, DEPTH_FORMAT_LAST }

  List available file formats for saving depth maps.
- enum POINT_CLOUD_FORMAT {
  POINT_CLOUD_FORMAT_XYZ_ASCII, POINT_CLOUD_FORMAT_PCD_ASCII, POINT_CLOUD_↩
  FORMAT_PLY_ASCII, POINT_CLOUD_FORMAT_VTK_ASCII,
  POINT_CLOUD_FORMAT_LAST }

  List available file formats for saving point clouds. Stores the spatial coordinates (x,y,z) of each pixel and optionally its RGB color.
- enum TRACKING_STATE {
  TRACKING_STATE_SEARCHING, TRACKING_STATE_OK, TRACKING_STATE_OFF, TRACKING↩
  _STATE_FPS_TOO_LOW,
  TRACKING_STATE_LAST }

  List the different states of positional tracking.
- enum AREA_EXPORT_STATE {
  AREA_EXPORT_STATE_SUCCESS, AREA_EXPORT_STATE_RUNNING, AREA_EXPORT_STATE_↩
  NOT_STARTED, AREA_EXPORT_STATE_FILE_EMPTY,
  AREA_EXPORT_STATE_FILE_ERROR, AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED,
  AREA_EXPORT_STATE_LAST }

  List the different states of spatial memory area export.
- enum REFERENCE_FRAME { REFERENCE_FRAME_WORLD, REFERENCE_FRAME_CAMERA, RE↩
  FERENCE_FRAME_LAST }

Define which type of position matrix is used to store camera path and pose.

- enum SPATIAL_MAPPING_STATE { SPATIAL_MAPPING_STATE_INITIALIZING, SPATIAL_MAPPING_STATE_OK, SPATIAL_MAPPIN↩ G_STATE_NOT_ENOUGH_MEMORY, SPATIAL_MAPPING_STATE_NOT_ENABLED, SPATIAL_MAPPING_STATE_FPS_TOO_LOW, SPATIAL_MAPPING_STATE_LAST }

  Gives the spatial mapping state.
- enum SVO_COMPRESSION_MODE { SVO_COMPRESSION_MODE_RAW, SVO_COMPRESSION↩ _MODE_LOSSLESS, SVO_COMPRESSION_MODE_LOSSY, SVO_COMPRESSION_MODE_LAST }

  List available compression modes for SVO recording.

Detailed Description

Enumeration Type Documentation

ERROR_CODE

`enum ERROR_CODE`

List error codes in the ZED SDK.

Enumerator

SUCCESS	Standard code for successful behavior.
ERROR_CODE_FAILURE	Standard code for unsuccessful behavior.
ERROR_CODE_NO_GPU_COMPATIBLE	No GPU found or CUDA capability of the device is not supported.
ERROR_CODE_NOT_ENOUGH_GPUMEM	Not enough GPU memory for this depth mode please try a faster mode (such as PERFORMANCE mode).
ERROR_CODE_CAMERA_NOT_DETECTED	The ZED camera is not plugged or detected.
ERROR_CODE_INVALID_RESOLUTION	For Nvidia Jetson X1 only, resolution not yet supported (USB3.0 bandwidth).
ERROR_CODE_LOW_USB_BANDWIDTH	This issue can occurs when you use multiple ZED or a USB 2.0 port (bandwidth issue).
ERROR_CODE_CALIBRATION_FILE_NOT_AV↩ AILABLE	ZED calibration file is not found on the host machine. Use ZED Explorer or ZED Calibration to get one.
ERROR_CODE_INVALID_SVO_FILE	The provided SVO file is not valid.
ERROR_CODE_SVO_RECORDING_ERROR	An recorder related error occurred (not enough free storage, invalid file).
ERROR_CODE_INVALID_COORDINATE_SYS↩ TEM	The requested coordinate system is not available.
ERROR_CODE_INVALID_FIRMWARE	The firmware of the ZED is out of date. Update to the latest version.
ERROR_CODE_INVALID_FUNCTION_PARA↩ METERS	An invalid parameter has been set for the function.
ERROR_CODE_NOT_A_NEW_FRAME	in grab() only, the current call return the same frame as last call. Not a new frame.
ERROR_CODE_CUDA_ERROR	in grab() only, a CUDA error has been detected in the process. Activate verbose in sl::Camera::open for more info.
ERROR_CODE_CAMERA_NOT_INITIALIZED	in grab() only, ZED SDK is not initialized. Probably a missing call to sl::Camera::open.

Enumerator

ERROR_CODE_NVIDIA_DRIVER_OUT_OF_D↩ ATE	your NVIDIA driver is too old and not compatible with your current CUDA version.
ERROR_CODE_INVALID_FUNCTION_CALL	the call of the function is not valid in the current context. Could be a missing call of sl::Camera::open.
ERROR_CODE_CORRUPTED_SDK_INSTALLA↩ TION	The SDK wasn't able to load its dependencies, the installer should be launched.
ERROR_CODE_LAST	

MESH_FILE_FORMAT

`enum MESH_FILE_FORMAT`

List available mesh file formats.

Enumerator

MESH_FILE_PLY	Contains only vertices and faces.
MESH_FILE_PLY_BIN	Contains only vertices and faces, encoded in binary.
MESH_FILE_OBJ	Contains vertices, normals, faces and textures informations if possible.
MESH_FILE_LAST	

MESH_TEXTURE_FORMAT

`enum MESH_TEXTURE_FORMAT`

List available mesh texture formats.

Enumerator

MESH_TEXTURE_RGB	The texture has 3 channels.
MESH_TEXTURE_RGBA	The texture has 4 channels.
MESH_TEXTURE_LAST	

FILTER

`enum FILTER`

List available mesh filtering intensity.

Enumerator

FILTER_LOW	Soft decimation and smoothing.
FILTER_MEDIUM	Decimate the number of faces and apply a soft smooth.
FILTER_HIGH	Drasticly reduce the number of faces.

RESOLUTION [1/2]

`enum RESOLUTION`

List the spatial mapping resolution presets.

Enumerator

RESOLUTION_HIGH	Create a detail geometry, requires lots of memory.
RESOLUTION_MEDIUM	Smalls variations in the geometry will disappear, useful for big object
RESOLUTION_LOW	Keeps only huge variations of the geometry , useful outdoor.

RANGE

`enum RANGE`

List the spatial mapping depth range presets.

Enumerator

RANGE_NEAR	Only depth close to the camera will be used by the spatial mapping.
RANGE_MEDIUM	Medium depth range.
RANGE_FAR	Takes into account objects that are far, useful outdoor.

MEM

`enum MEM`

List available memory type.

Enumerator

MEM_CPU	CPU Memory (Processor side).
MEM_GPU	GPU Memory (Graphic card side).

COPY_TYPE

`enum COPY_TYPE`

List available copy operation on Mat.

Enumerator

COPY_TYPE_CPU_CPU	copy data from CPU to CPU.
COPY_TYPE_CPU_GPU	copy data from CPU to GPU.
COPY_TYPE_GPU_GPU	copy data from GPU to GPU.
COPY_TYPE_GPU_CPU	copy data from GPU to CPU.

MAT_TYPE

`enum MAT_TYPE`

List available Mat formats.

Enumerator

MAT_TYPE_32F_C1	float 1 channel.
MAT_TYPE_32F_C2	float 2 channels.
MAT_TYPE_32F_C3	float 3 channels.

Enumerator

MAT_TYPE_32F_C4	float 4 channels.
MAT_TYPE_8U_C1	unsigned char 1 channel.
MAT_TYPE_8U_C2	unsigned char 2 channels.
MAT_TYPE_8U_C3	unsigned char 3 channels.
MAT_TYPE_8U_C4	unsigned char 4 channels.

RESOLUTION [2/2]

`enum RESOLUTION`

Represents the available resolution defined in sl::cameraResolution.

Note

Since v1.0, RESOLUTION_VGA mode has been updated to WVGA (from 640∗480 to 672∗376) and requires a firmware update to function (>= 1142). Firmware can be updated in the ZED Explorer.

Warning

NVIDIA Jetson X1 only supports RESOLUTION_HD1080@15, RESOLUTION_HD720@30/15, and RESOLUTION_VGA@60/30/15.

Enumerator

RESOLUTION_HD2K	2208∗1242, available framerates: 15 fps.
RESOLUTION_HD1080	1920∗1080, available framerates: 15, 30 fps.
RESOLUTION_HD720	1280∗720, available framerates: 15, 30, 60 fps.
RESOLUTION_VGA	672∗376, available framerates: 15, 30, 60, 100 fps.
RESOLUTION_LAST	

CAMERA_SETTINGS

`enum CAMERA_SETTINGS`

List available camera settings for the ZED camera (contrast, hue, saturation, gain...).

Each enum defines one of those settings.

Enumerator

CAMERA_SETTINGS_BRIGHTNESS	Defines the brightness control. Affected value should be between 0 and 8.
CAMERA_SETTINGS_CONTRAST	Defines the contrast control. Affected value should be between 0 and 8.
CAMERA_SETTINGS_HUE	Defines the hue control. Affected value should be between 0 and 11.
CAMERA_SETTINGS_SATURATION	Defines the saturation control. Affected value should be between 0 and 8.
CAMERA_SETTINGS_GAIN	Defines the gain control. Affected value should be between 0 and 100 for manual control. If ZED_EXPOSURE is set to -1, the gain is in auto mode too.

Enumerator

CAMERA_SETTINGS_EXPOSURE	Defines the exposure control. A -1 value enable the AutoExposure/AutoGain control,as the boolean parameter (default) does. Affected value should be between 0 and 100 for manual control.
CAMERA_SETTINGS_WHITEBALANCE	Defines the color temperature control. Affected value should be between 2800 and 6500 with a step of 100. A value of -1 set the AWB (auto white balance), as the boolean parameter (default) does.
CAMERA_SETTINGS_AUTO_WHITEBALANCE	Defines the status of white balance (automatic or manual). A value of 0 disable the AWB, while 1 activate it.
CAMERA_SETTINGS_LAST	

SELF_CALIBRATION_STATE

`enum SELF_CALIBRATION_STATE`

Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the sl::Camera↩ ::open or Reset function.

You can follow the current status for the self-calibration any time once ZED object has been construct.

Enumerator

SELF_CALIBRATION_STATE_NOT_STARTED	Self calibration has not run yet (no sl::Camera::open or sl::Camera::resetSelfCalibration called).
SELF_CALIBRATION_STATE_RUNNING	Self calibration is currently running.
SELF_CALIBRATION_STATE_FAILED	Self calibration has finished running but did not manage to get accurate values. Old parameters are taken instead.
SELF_CALIBRATION_STATE_SUCCESS	Self calibration has finished running and did manage to get accurate values. New parameters are set.
SELF_CALIBRATION_STATE_LAST	

DEPTH_MODE

`enum DEPTH_MODE`

List available depth computation modes.

Enumerator

DEPTH_MODE_NONE	This mode does not compute any depth map. Only rectified stereo images will be available.
DEPTH_MODE_PERFORMANCE	Fastest mode for depth computation.
DEPTH_MODE_MEDIUM	Balanced quality mode. Depth map is robust in any environment and requires medium resources for computation.
DEPTH_MODE_QUALITY	Best quality mode. Requires more compute power.
DEPTH_MODE_LAST	

SENSING_MODE

`enum SENSING_MODE`

List available depth sensing modes.

Enumerator

SENSING_MODE_STANDARD	This mode outputs ZED standard depth map that preserves edges and depth accuracy. Applications example: Obstacle detection, Automated navigation, People detection, 3D reconstruction.
SENSING_MODE_FILL	This mode outputs a smooth and fully dense depth map. Applications example: AR/VR, Mixed-reality capture, Image post-processing.
SENSING_MODE_LAST	

UNIT

`enum UNIT`

List available unit for measures.

Enumerator

UNIT_MILLIMETER	International System, 1/1000 METER.
UNIT_CENTIMETER	International System, 1/100 METER.
UNIT_METER	International System, 1 METER
UNIT_INCH	Imperial Unit, 1/12 FOOT
UNIT_FOOT	Imperial Unit, 1 FOOT
UNIT_LAST	

COORDINATE_SYSTEM

`enum COORDINATE_SYSTEM`

List available coordinates systems for positional tracking and 3D measures.

Enumerator

COORDINATE_SYSTEM_IMAGE	Standard coordinates system in computer vision. Used in OpenCV : see here : http://docs.opencv.org/2.↩ 4/modules/calib3d/doc/camera_↩ calibration_and_3d_reconstruction.↩ html
COORDINATE_SYSTEM_LEFT_HANDED_Y_UP	Left-Handed with Y up and Z forward. Used in Unity with DirectX.
COORDINATE_SYSTEM_RIGHT_HANDED_Y_↩ UP	Right-Handed with Y pointing up and Z backward. Used in OpenGL.
COORDINATE_SYSTEM_RIGHT_HANDED_Z_↩ UP	Right-Handed with Z pointing up and Y forward. Used in 3DSMax.
COORDINATE_SYSTEM_LEFT_HANDED_Z_UP	Left-Handed with Z axis pointing up and X forward. Used in Unreal Engine.
COORDINATE_SYSTEM_LAST	

MEASURE

`enum MEASURE`

List retrievable measures.

Enumerator

MEASURE_DISPARITY	Disparity map, sl::MAT_TYPE_32F_C1.
MEASURE_DEPTH	Depth map, sl::MAT_TYPE_32F_C1.
MEASURE_CONFIDENCE	Certainty/confidence of the disparity map, sl::MAT_TYPE_32F_C1.
MEASURE_XYZ	Point cloud, sl::MAT_TYPE_32F_C4, channel 4 is empty.
MEASURE_XYZRGBA	Colored point cloud, sl::MAT_TYPE_32F_C4, channel 4 contains color in R-G-B-A order.
MEASURE_XYZBGRA	Colored point cloud, sl::MAT_TYPE_32F_C4, channel 4 contains color in B-G-R-A order.
MEASURE_XYZARGB	Colored point cloud, sl::MAT_TYPE_32F_C4, channel 4 contains color in A-R-G-B order.
MEASURE_XYZABGR	Colored point cloud, sl::MAT_TYPE_32F_C4, channel 4 contains color in A-B-G-R order.
MEASURE_NORMALS	Normals vector, sl::MAT_TYPE_32F_C4, channel 4 is empty (set to 0)
MEASURE_DISPARITY_RIGHT	Disparity map for right sensor, sl::MAT_TYPE_32F_C1.
MEASURE_DEPTH_RIGHT	Depth map for right sensor, sl::MAT_TYPE_32F_C1.
MEASURE_XYZ_RIGHT	Point cloud for right sensor, sl::MAT_TYPE_32F_C1, channel 4 is empty.
MEASURE_XYZRGBA_RIGHT	Colored point cloud for right sensor, sl::MAT_TYPE_32F_C4, channel 4 contains color in R-G-B-A order.
MEASURE_XYZBGRA_RIGHT	Colored point cloud for right sensor, sl::MAT_TYPE_32F_C4, channel 4 contains color in B-G-R-A order.
MEASURE_XYZARGB_RIGHT	Colored point cloud for right sensor, sl::MAT_TYPE_32F_C4, channel 4 contains color in A-R-G-B order.
MEASURE_XYZABGR_RIGHT	Colored point cloud for right sensor, sl::MAT_TYPE_32F_C4, channel 4 contains color in A-B-G-R order.
MEASURE_NORMALS_RIGHT	Normals vector for right view, sl::MAT_TYPE_32F_C4, channel 4 is empty (set to 0)
MEASURE_LAST	

VIEW

`enum VIEW`

List available views.

Enumerator

VIEW_LEFT	Left RGBA image, sl::MAT_TYPE_8U_C4.
VIEW_RIGHT	Right RGBA image, sl::MAT_TYPE_8U_C4.
VIEW_LEFT_GRAY	Left GRAY image, sl::MAT_TYPE_8U_C1.
VIEW_RIGHT_GRAY	Right GRAY image, sl::MAT_TYPE_8U_C1.
VIEW_LEFT_UNRECTIFIED	Left RGBA unrectified image, sl::MAT_TYPE_8U_C4.
VIEW_RIGHT_UNRECTIFIED	Right RGBA unrectified image, sl::MAT_TYPE_8U_C4.
VIEW_LEFT_UNRECTIFIED_GRAY	Left GRAY unrectified image, sl::MAT_TYPE_8U_C1.
VIEW_RIGHT_UNRECTIFIED_GRAY	Right GRAY unrectified image, sl::MAT_TYPE_8U_C1.

Enumerator

VIEW_SIDE_BY_SIDE	Left and right image (the image width is therefore doubled). RGBA image, sl::MAT_TYPE_8U_C4.
VIEW_DEPTH	Color rendering of the depth, sl::MAT_TYPE_8U_C4.
VIEW_CONFIDENCE	Color rendering of the depth confidence, sl::MAT_TYPE_8U_C4.
VIEW_NORMALS	Color rendering of the normals, sl::MAT_TYPE_8U_C4.
VIEW_DEPTH_RIGHT	Color rendering of the right depth mapped on right sensor, sl::MAT_TYPE_8U_C4.
VIEW_NORMALS_RIGHT	Color rendering of the normals mapped on right sensor, sl::MAT_TYPE_8U_C4.
VIEW_LAST	

DEPTH_FORMAT

`enum DEPTH_FORMAT`

List available file formats for saving depth maps.

Enumerator

DEPTH_FORMAT_PNG	PNG image format in 16bits. 32bits depth is mapped to 16bits color image to preserve the consistency of the data range.
DEPTH_FORMAT_PFM	stream of bytes, graphic image file format.
DEPTH_FORMAT_PGM	gray-scale image format.
DEPTH_FORMAT_LAST	

POINT_CLOUD_FORMAT

`enum POINT_CLOUD_FORMAT`

List available file formats for saving point clouds. Stores the spatial coordinates (x,y,z) of each pixel and optionally its RGB color.

Enumerator

POINT_CLOUD_FORMAT_XYZ_ASCII	Generic point cloud file format, without color information.
POINT_CLOUD_FORMAT_PCD_ASCII	Point Cloud Data file, with color information.
POINT_CLOUD_FORMAT_PLY_ASCII	PoLYgon file format, with color information.
POINT_CLOUD_FORMAT_VTK_ASCII	Visualization ToolKit file, without color information.
POINT_CLOUD_FORMAT_LAST	

TRACKING_STATE

`enum TRACKING_STATE`

List the different states of positional tracking.

Enumerator

TRACKING_STATE_SEARCHING	The camera is searching for a previously known position to locate itself.
TRACKING_STATE_OK	Positional tracking is working normally.
TRACKING_STATE_OFF	Positional tracking is not enabled.
TRACKING_STATE_FPS_TOO_LOW	Effective FPS is too low to give proper results for motion tracking. Consider using PERFORMANCES parameters (DEPTH_MODE_PERFORMANCE, low camera resolution (VGA,HD720))
TRACKING_STATE_LAST	

AREA_EXPORT_STATE

`enum AREA_EXPORT_STATE`

List the different states of spatial memory area export.

Enumerator

AREA_EXPORT_STATE_SUCCESS	The spatial memory file has been successfully created.
AREA_EXPORT_STATE_RUNNING	The spatial memory is currently written.
AREA_EXPORT_STATE_NOT_STARTED	The spatial memory file exportation has not been called.
AREA_EXPORT_STATE_FILE_EMPTY	The spatial memory contains no data, the file is empty.
AREA_EXPORT_STATE_FILE_ERROR	The spatial memory file has not been written because of a wrong file name.
AREA_EXPORT_STATE_SPATIAL_MEMORY_↩ DISABLED	The spatial memory learning is disable, no file can be created.
AREA_EXPORT_STATE_LAST	

REFERENCE_FRAME

`enum REFERENCE_FRAME`

Define which type of position matrix is used to store camera path and pose.

Enumerator

REFERENCE_FRAME_WORLD	The transform of sl::Pose will contains the motion with reference to the world frame (previously called PATH).
REFERENCE_FRAME_CAMERA	The transform of sl::Pose will contains the motion with reference to the previous camera frame (previously called POSE).
REFERENCE_FRAME_LAST	

SPATIAL_MAPPING_STATE

`enum SPATIAL_MAPPING_STATE`

Gives the spatial mapping state.

Enumerator

SPATIAL_MAPPING_STATE_INITIALIZING	The spatial mapping is initializing.
SPATIAL_MAPPING_STATE_OK	The depth and tracking data were correctly integrated in the fusion algorithm.
SPATIAL_MAPPING_STATE_NOT_ENOUGH_↩ MEMORY	The maximum memory dedicated to the scanning has been reach, the mesh will no longer be updated.
SPATIAL_MAPPING_STATE_NOT_ENABLED	Camera::enableSpatialMapping() wasn't called (or the scanning was stopped and not relaunched).
SPATIAL_MAPPING_STATE_FPS_TOO_LOW	Effective FPS is too low to give proper results for spatial mapping. Consider using PERFORMANCES parameters (DEPTH_MODE_PERFORMANCE, low camera resolution (VGA,HD720), spatial mapping low resolution)
SPATIAL_MAPPING_STATE_LAST	

SVO_COMPRESSION_MODE

`enum SVO_COMPRESSION_MODE`

List available compression modes for SVO recording.

sl::SVO_COMPRESSION_MODE_LOSSLESS is an improvement of previous lossless compression (used in ZED Explorer), even if size may be bigger, compression time is much faster.

Enumerator

SVO_COMPRESSION_MODE_RAW	RAW images, no compression.
SVO_COMPRESSION_MODE_LOSSLESS	new Lossless, with PNG/ZSTD based compression : avg size = 42% of RAW).
SVO_COMPRESSION_MODE_LOSSY	new Lossy, with JPEG based compression : avg size = 22% of RAW).
SVO_COMPRESSION_MODE_LAST	

Namespace Documentation

sl Namespace Reference

Classes

- struct CalibrationParameters

  Intrinsic parameters of each cameras and extrinsic (translation and rotation).
- class Camera

  The main class to use the ZED camera.
- struct CameraInformation

  Camera specific parameters.
- struct CameraParameters

  Intrinsic parameters of a camera.
- class Chunk

  Represents a sub mesh, it contains local vertices and triangles.
- class InitParameters

  Parameters that will be fixed for the whole execution life time of the sl::Camera.
- class Mat

  The Mat class can handle multiple matrix format from 1 to 4 channels, with different value types (float or uchar), and can be stored CPU and/or GPU side.
- class Matrix3f

  Represents a generic three-dimensional matrix.
- class Matrix4f

  Represents a generic fourth-dimensional matrix.
- class Mesh

  A mesh contains the geometric (and optionally texturing) data of the scene computed by the spatial mapping.
- class MeshFilterParameters

  Parameters for the optional filtering step of a sl::Mesh.
  A default constructor is enabled and set to its default parameters.
- class Orientation

  Designed to contain orientation (quaternion) data of the positional tracking.
- class Pose

  Contains the positional tracking data which gives the position and orientation of the ZED in 3D space.
- struct RecordingState

  Recording structure that contains information about SVO.
- struct Resolution

  Width and height of an array.
- class Rotation

  Designed to contain rotation data of the positional tracking. It inherits from the generic sl::Matrix3f.
- class RuntimeParameters

  Parameters that defines the behavior of the sl::Camera::grab().
- class SpatialMappingParameters

  Sets the spatial mapping parameters.
- class String

  Defines a string.
- class Texture

  Contains information about texture image associated to a sl::Mesh.
- class TrackingParameters

Parameters for ZED tracking initialization.

- class Transform

  Designed to contain translation and rotation data of the positional tracking.
- class Translation

  Designed to contain translation data of the positional tracking.
- class Vector2

  Represents a two dimensions vector for both CPU and GPU.
- class Vector3

  Represents a three dimensions vector for both CPU and GPU.
- class Vector4

  Represents a four dimensions vector for both CPU and GPU.

Typedefs

Types definition

- typedef float float1
- typedef Vector2< float > float2
- typedef Vector3< float > float3
- typedef Vector4< float > float4
- typedef unsigned char uchar1
- typedef Vector2< unsigned char > uchar2
- typedef Vector3< unsigned char > uchar3
- typedef Vector4< unsigned char > uchar4
- typedef double double1
- typedef Vector2< double > double2
- typedef Vector3< double > double3
- typedef Vector4< double > double4
- typedef unsigned int uint1
- typedef Vector2< unsigned int > uint2
- typedef Vector3< unsigned int > uint3
- typedef Vector4< unsigned int > uint4
- typedef unsigned long long timeStamp

Enumerations

- enum ERROR_CODE {
  SUCCESS, ERROR_CODE_FAILURE, ERROR_CODE_NO_GPU_COMPATIBLE, ERROR_CODE_N↩
  OT_ENOUGH_GPUMEM,
  ERROR_CODE_CAMERA_NOT_DETECTED, ERROR_CODE_INVALID_RESOLUTION, ERROR_C↩
  ODE_LOW_USB_BANDWIDTH, ERROR_CODE_CALIBRATION_FILE_NOT_AVAILABLE,
  ERROR_CODE_INVALID_SVO_FILE, ERROR_CODE_SVO_RECORDING_ERROR, ERROR_CODE↩
  _INVALID_COORDINATE_SYSTEM, ERROR_CODE_INVALID_FIRMWARE,
  ERROR_CODE_INVALID_FUNCTION_PARAMETERS, ERROR_CODE_NOT_A_NEW_FRAME, ER↩
  ROR_CODE_CUDA_ERROR, ERROR_CODE_CAMERA_NOT_INITIALIZED,
  ERROR_CODE_NVIDIA_DRIVER_OUT_OF_DATE, ERROR_CODE_INVALID_FUNCTION_CALL,
  ERROR_CODE_CORRUPTED_SDK_INSTALLATION, ERROR_CODE_LAST }

  List error codes in the ZED SDK.
- enum MESH_FILE_FORMAT { MESH_FILE_PLY, MESH_FILE_PLY_BIN, MESH_FILE_OBJ, MESH_F↩
  ILE_LAST }

  List available mesh file formats.
- enum MESH_TEXTURE_FORMAT { MESH_TEXTURE_RGB, MESH_TEXTURE_RGBA, MESH_TEX↩
  TURE_LAST }

  List available mesh texture formats.
- enum MEM { MEM_CPU = 1, MEM_GPU = 2 }

List available memory type.

- enum COPY_TYPE { COPY_TYPE_CPU_CPU, COPY_TYPE_CPU_GPU, COPY_TYPE_GPU_GPU, C↩ OPY_TYPE_GPU_CPU }

  List available copy operation on Mat.
- enum MAT_TYPE {
  MAT_TYPE_32F_C1, MAT_TYPE_32F_C2, MAT_TYPE_32F_C3, MAT_TYPE_32F_C4,
  MAT_TYPE_8U_C1, MAT_TYPE_8U_C2, MAT_TYPE_8U_C3, MAT_TYPE_8U_C4 }

  List available Mat formats.
- enum RESOLUTION {
  RESOLUTION_HD2K, RESOLUTION_HD1080, RESOLUTION_HD720, RESOLUTION_VGA,
  RESOLUTION_LAST }

  Represents the available resolution defined in sl::cameraResolution.
- enum CAMERA_SETTINGS {
  CAMERA_SETTINGS_BRIGHTNESS, CAMERA_SETTINGS_CONTRAST, CAMERA_SETTINGS_H↩ UE, CAMERA_SETTINGS_SATURATION,
  CAMERA_SETTINGS_GAIN, CAMERA_SETTINGS_EXPOSURE, CAMERA_SETTINGS_WHITEBA↩ LANCE, CAMERA_SETTINGS_AUTO_WHITEBALANCE,
  CAMERA_SETTINGS_LAST }

  List available camera settings for the ZED camera (contrast, hue, saturation, gain...).
- enum SELF_CALIBRATION_STATE {
  SELF_CALIBRATION_STATE_NOT_STARTED, SELF_CALIBRATION_STATE_RUNNING, SELF_C↩ ALIBRATION_STATE_FAILED, SELF_CALIBRATION_STATE_SUCCESS,
  SELF_CALIBRATION_STATE_LAST }

  Status for self calibration. Since v0.9.3, self-calibration is done in background and start in the sl::Camera::open or Reset function.
- enum DEPTH_MODE {
  DEPTH_MODE_NONE, DEPTH_MODE_PERFORMANCE, DEPTH_MODE_MEDIUM, DEPTH_M↩ ODE_QUALITY,
  DEPTH_MODE_LAST }

  List available depth computation modes.
- enum SENSING_MODE { SENSING_MODE_STANDARD, SENSING_MODE_FILL, SENSING_MOD↩ E_LAST }

  List available depth sensing modes.
- enum UNIT {
  UNIT_MILLIMETER, UNIT_CENTIMETER, UNIT_METER, UNIT_INCH,
  UNIT_FOOT, UNIT_LAST }

  List available unit for measures.
- enum COORDINATE_SYSTEM {
  COORDINATE_SYSTEM_IMAGE, COORDINATE_SYSTEM_LEFT_HANDED_Y_UP, COORDINAT↩ E_SYSTEM_RIGHT_HANDED_Y_UP, COORDINATE_SYSTEM_RIGHT_HANDED_Z_UP,
  COORDINATE_SYSTEM_LEFT_HANDED_Z_UP, COORDINATE_SYSTEM_LAST }

  List available coordinates systems for positional tracking and 3D measures.
- enum MEASURE {
  MEASURE_DISPARITY, MEASURE_DEPTH, MEASURE_CONFIDENCE, MEASURE_XYZ,
  MEASURE_XYZRGBA, MEASURE_XYZBGRA, MEASURE_XYZARGB, MEASURE_XYZABGR,
  MEASURE_NORMALS, MEASURE_DISPARITY_RIGHT, MEASURE_DEPTH_RIGHT, MEASURE_X↩ YZ_RIGHT,
  MEASURE_XYZRGBA_RIGHT, MEASURE_XYZBGRA_RIGHT, MEASURE_XYZARGB_RIGHT, ME↩ ASURE_XYZABGR_RIGHT,
  MEASURE_NORMALS_RIGHT, MEASURE_LAST }

  List retrievable measures.
- enum VIEW {
  VIEW_LEFT, VIEW_RIGHT, VIEW_LEFT_GRAY, VIEW_RIGHT_GRAY,
  VIEW_LEFT_UNRECTIFIED, VIEW_RIGHT_UNRECTIFIED, VIEW_LEFT_UNRECTIFIED_GRAY, V↩ IEW_RIGHT_UNRECTIFIED_GRAY,

VIEW_SIDE_BY_SIDE, VIEW_DEPTH, VIEW_CONFIDENCE, VIEW_NORMALS, VIEW_DEPTH_RIGHT, VIEW_NORMALS_RIGHT, VIEW_LAST }

List available views.

- enum DEPTH_FORMAT { DEPTH_FORMAT_PNG, DEPTH_FORMAT_PFM, DEPTH_FORMAT_PGM, DEPTH_FORMAT_LAST }

  List available file formats for saving depth maps.
- enum POINT_CLOUD_FORMAT { POINT_CLOUD_FORMAT_XYZ_ASCII, POINT_CLOUD_FORMAT_PCD_ASCII, POINT_CLOUD_FORMAT_PLY_ASCII, POINT_CLOUD_FORMAT_VTK_ASCII, POINT_CLOUD_FORMAT_LAST }

  List available file formats for saving point clouds. Stores the spatial coordinates (x,y,z) of each pixel and optionally its RGB color.
- enum TRACKING_STATE { TRACKING_STATE_SEARCHING, TRACKING_STATE_OK, TRACKING_STATE_OFF, TRACKING_STATE_FPS_TOO_LOW, TRACKING_STATE_LAST }

  List the different states of positional tracking.
- enum AREA_EXPORT_STATE { AREA_EXPORT_STATE_SUCCESS, AREA_EXPORT_STATE_RUNNING, AREA_EXPORT_STATE_NOT_STARTED, AREA_EXPORT_STATE_FILE_EMPTY, AREA_EXPORT_STATE_FILE_ERROR, AREA_EXPORT_STATE_SPATIAL_MEMORY_DISABLED, AREA_EXPORT_STATE_LAST }

  List the different states of spatial memory area export.
- enum REFERENCE_FRAME { REFERENCE_FRAME_WORLD, REFERENCE_FRAME_CAMERA, REFERENCE_FRAME_LAST }

  Define which type of position matrix is used to store camera path and pose.
- enum SPATIAL_MAPPING_STATE { SPATIAL_MAPPING_STATE_INITIALIZING, SPATIAL_MAPPING_STATE_OK, SPATIAL_MAPPING_STATE_NOT_ENOUGH_MEMORY, SPATIAL_MAPPING_STATE_NOT_ENABLED, SPATIAL_MAPPING_STATE_FPS_TOO_LOW, SPATIAL_MAPPING_STATE_LAST }

  Gives the spatial mapping state.
- enum SVO_COMPRESSION_MODE { SVO_COMPRESSION_MODE_RAW, SVO_COMPRESSION_MODE_LOSSLESS, SVO_COMPRESSION_MODE_LOSSY, SVO_COMPRESSION_MODE_LAST }

  List available compression modes for SVO recording.

Functions

- void sleep_ms (int time)

  Tells the program to wait for x ms.
- SL_SDK_EXPORT bool saveDepthAs (sl::Camera &zed, sl::DEPTH_FORMAT format, sl::String name, float factor=1.)

  Writes the current depth map into a file.
- SL_SDK_EXPORT bool saveDepthAs (sl::Mat &depth, sl::DEPTH_FORMAT format, sl::String name, float factor=1.)

  Writes the current depth map into a file.
- SL_SDK_EXPORT bool savePointCloudAs (sl::Camera &zed, sl::POINT_CLOUD_FORMAT format, sl::String name, bool with_color=false, bool keep_occluded_point=false)

  Writes the current point cloud into a file.
- SL_SDK_EXPORT bool savePointCloudAs (sl::Mat &cloud, sl::POINT_CLOUD_FORMAT format, sl::String name, bool with_color=false, bool keep_occluded_point=false)

  Writes the current point cloud into a file.
- static timeStamp getCurrentTimeStamp ()

Returns the current timestamp at the time the function is called. Can be compared to sl::Camera::getCamera↩ Timestamp for synchronization.

Enumeration conversion

- static std::string errorCode2str (ERROR_CODE err)
  Converts the given ERROR_CODE into a string.
- static std::string resolution2str (RESOLUTION res)
  Converts the given RESOLUTION into a string.
- static std::string statusCode2str (SELF_CALIBRATION_STATE state)
  Converts the given SELF_CALIBRATION_STATE into a string.
- static DEPTH_MODE str2mode (std::string mode)
  Converts the given string into a DEPTH_MODE.
- static std::string depthMode2str (DEPTH_MODE mode)
  Converts the given DEPTH_MODE into a string.
- static std::string sensingMode2str (SENSING_MODE mode)
  Converts the given SENSING_MODE into a string.
- static std::string unit2str (UNIT unit)
  Converts the given UNIT into a string.
- static UNIT str2unit (std::string unit)
  Converts the given string into a UNIT.
- static std::string trackingState2str (TRACKING_STATE state)
  Converts the given TRACKING_STATE into a string.
- static std::string spatialMappingState2str (SPATIAL_MAPPING_STATE state)
  Converts the given SPATIAL_MAPPING_STATE into a string.

Variables

Unavailable Values

- static const float TOO_FAR = INFINITY
- static const float TOO_CLOSE = -INFINITY
- static const float OCCLUSION_VALUE = NAN

ZED Camera Resolution

- static const std::vector< std::pair< int, int > > cameraResolution

Typedef Documentation

float1

```
typedef float float1
```

float2

```
typedef Vector2<float> float2
```

float3

```
typedef Vector3<float> float3
```

float4

```
typedef Vector4<float> float4
```

uchar1

```
typedef unsigned char uchar1
```

uchar2

```
typedef Vector2<unsigned char> uchar2
```

uchar3

```
typedef Vector3<unsigned char> uchar3
```

uchar4

```
typedef Vector4<unsigned char> uchar4
```

double1

```
typedef double double1
```

double2

```
typedef Vector2<double> double2
```

double3

```
typedef Vector3<double> double3
```

double4

```
typedef Vector4<double> double4
```

uint1

```
typedef unsigned int uint1
```

uint2

```
typedef Vector2<unsigned int> uint2
```

uint3

```
typedef Vector3<unsigned int> uint3
```

uint4

```
typedef Vector4<unsigned int> uint4
```

timeStamp

```
typedef unsigned long long timeStamp
```

Variable Documentation

TOO_FAR

```
const float TOO_FAR = INFINITY  [static]
```

Defines an unavailable depth value that is above the depth Max value.

TOO_CLOSE

```
const float TOO_CLOSE = -INFINITY  [static]
```

Defines an unavailable depth value that is below the depth Min value.

OCCLUSION_VALUE

```
const float OCCLUSION_VALUE = NAN  [static]
```

Defines an unavailable depth value that is on an occluded image area.

cameraResolution

```
const std::vector<std::pair<int, int> > cameraResolution  [static]
```

Initial value:

```
= {
        std::make_pair(2208, 1242),
        std::make_pair(1920, 1080),
        std::make_pair(1280, 720),
        std::make_pair(672, 376)
    }
```

Available video modes for the ZED camera.

Class Documentation

CalibrationParameters Struct Reference

Intrinsic parameters of each cameras and extrinsic (translation and rotation).

Public Attributes

- sl::float3 R
- sl::float3 T
- CameraParameters left_cam
- CameraParameters right_cam

Detailed Description

Intrinsic parameters of each cameras and extrinsic (translation and rotation).

Note

The calibration/rectification process, called during sl::Camera::open, is using the raw parameters defined in the SNXXX.conf file, where XXX is the ZED Serial Number.
Those values may be adjusted or not by the Self-Calibration to get a proper image alignment. After sl::Camera::open is done (with or without Self-Calibration activated) success, most of the stereo parameters (except Baseline of course) should be 0 or very close to 0.
It means that images after rectification process (given by retrieveImage()) are aligned as if they were taken by a "perfect" stereo camera, defined by the new CalibrationParameters.

Member Data Documentation

R

```
sl::float3 R
```

Rotation (using Rodrigues' transformation) between the two sensors. Defined as 'tilt', 'convergence' and 'roll'.

T

```
sl::float3 T
```

Translation between the two sensors. T.x is the distance between the two cameras (baseline) in the sl::U↩ NIT chosen during sl::Camera::open (mm, cm, meters, inches...).

left_cam

```
CameraParameters left_cam
```

Intrinsic parameters of the left camera

right_cam

```
CameraParameters right_cam
```

Intrinsic parameters of the right camera

Camera Class Reference

The main class to use the ZED camera.

Public Member Functions

- Camera ()

  Default constructor which creates an empty Camera.
- ~Camera ()

  Camera destructor.
- ERROR_CODE open (InitParameters init_parameters=InitParameters())

  Opens the ZED camera in the desired mode (live/SVO), sets all the defined parameters, checks hardware requirements and launch internal self calibration.
- bool isOpened ()

  Tests if the camera is opened and running.
- void close ()

  Closes the camera and free the memory. Camera::open can then be called again to reset the camera if needed.
- ERROR_CODE grab (RuntimeParameters rt_parameters=RuntimeParameters())

  Grabs a new image, rectifies it and computes the depth map. This function is typically called in the main loop.
- ERROR_CODE retrieveImage (Mat &mat, VIEW view=VIEW_LEFT, MEM type=MEM_CPU, int width=0, int height=0)

  Retrieves the desired image (Left, colorized_depth ...).
- Resolution getResolution ()

  Returns the current image size.
- CUcontext getCUDAContext ()

  Returns the CUDA context used for all the computation.

Camera informations

- CameraInformation getCameraInformation (Resolution resizer=Resolution(0, 0))
  Returns camera informations (calibration parameters, serial number and current firmware version).
- int getCameraSettings (CAMERA_SETTINGS setting)
  Returns the current value to the corresponding sl::CAMERA_SETTINGS (Gain, brightness, hue, exposure...).
- void setCameraSettings (CAMERA_SETTINGS settings, int value, bool use_default=false)
  Sets the value to the corresponding sl::CAMERA_SETTINGS (Gain, brightness, hue, exposure...).
- float getCameraFPS ()
  Returns the current FPS of the camera.
- void setCameraFPS (int desired_fps)
  Sets a new frame rate for the camera, or the closest available frame rate.
- float getCurrentFPS ()
  Returns the current FPS of the application/callback.
  It is based on the difference of camera timestamps between two successful grab().
- timeStamp getCameraTimestamp ()
  Returns the timestamp at the time the frame has been extracted from USB stream. (should be called after a grab()).
- timeStamp getCurrentTimestamp ()
  Returns the current timestamp at the time the function is called. Can be compared to the camera getCamera↩ Timestamp for synchronization.
  Use this function to compare the current timestamp and the camera timestamp, since they have the same reference (Computer start time).
- unsigned int getFrameDroppedCount ()
  Returns the number of frame dropped since sl::Camera::grab has been called for the first time.
  Based on camera timestamp and FPS comparison.
- int getSVOPosition ()
  Returns the current position of the SVO file.
- void setSVOPosition (int frame_number)
  Sets the position of the SVO file to a desired frame.
- int getSVONumberOfFrames ()
  Returns the number of frames in the SVO file.

Depth

- ERROR_CODE retrieveMeasure (Mat &mat, MEASURE measure=MEASURE_DEPTH, MEM type=MEM_CPU, int width=0, int height=0)
  Retrieves the desired measure (disparity, depth, point cloud ...).
- float getDepthMaxRangeValue ()
  Returns the current maximum distance of depth/disparity estimation.
- void setDepthMaxRangeValue (float depth_max_range)
  Sets the maximum distance of depth/disparity estimation (all values after this limit will be reported as sl::TOO↩ _FAR value).
- float getDepthMinRangeValue ()
  Returns the closest measurable distance by the camera, according to the camera and the depth map parameters.
- int getConfidenceThreshold ()
  Returns the current confidence threshold value apply to the disparity map(and by extension the depth map).
- void setConfidenceThreshold (int conf_threshold_value)
  Sets a threshold for the disparity map confidence (and by extension the depth map).

Positional Tracking

- ERROR_CODE enableTracking (TrackingParameters tracking_parameters=TrackingParameters())
  Initializes and start the tracking processes.
- sl::TRACKING_STATE getPosition (sl::Pose &camera_pose, REFERENCE_FRAME reference_↩ frame=sl::REFERENCE_FRAME_WORLD)
  Fills the position of the camera frame in the world frame and return the current state of the Tracker.
- sl::AREA_EXPORT_STATE getAreaExportState ()
  Returns the state of the spatial memory export process.

- void disableTracking (sl::String area_file_path="")
  Disables motion tracking.
- ERROR_CODE resetTracking (sl::Transform &path)
  Resets the tracking, re-initializes the path with the transformation matrix given.

Spatial Mapping

- ERROR_CODE enableSpatialMapping (SpatialMappingParameters spatial_mapping_parameters=Spatial↩ MappingParameters())
  Initializes and starts the spatial mapping processes.
- SPATIAL_MAPPING_STATE getSpatialMappingState ()
  Returns the current spatial mapping state.
- void requestMeshAsync ()
  Starts the mesh generation process in a non blocking thread from the spatial mapping process.
- ERROR_CODE getMeshRequestStatusAsync ()
  Returns the mesh generation status, useful after calling requestMeshAsync to know if you can call sl::Camera↩ ::retrieveMeshAsync().
- ERROR_CODE retrieveMeshAsync (sl::Mesh &mesh)
  Retrieves the generated mesh after calling requestMeshAsync.
- ERROR_CODE extractWholeMesh (sl::Mesh &mesh)
  Extracts the current mesh from the spatial mapping process.
- void pauseSpatialMapping (bool status)
  Switches the pause status of the data integration mechanism for the spatial mapping.
- void disableSpatialMapping ()
  Disables the Spatial Mapping process. All the spatial mapping functions are disables, mesh cannot be retrieves after this call.

Self calibration

- SELF_CALIBRATION_STATE getSelfCalibrationState ()
  Returns the current status of the self-calibration.
- void resetSelfCalibration ()
  Resets the self camera calibration. This function can be called at any time AFTER the sl::Camera::open function has been called.

Recorder

- ERROR_CODE enableRecording (sl::String video_filename, SVO_COMPRESSION_MODE compression_mode=SVO_COMPRESSION_MODE_LOSSLESS)
  Creates a file for recording the current frames.
- sl::RecordingState record ()
  Records the current frame provided by grab() into the file.
- void disableRecording ()
  Disables the recording and closes the generated file.

Static Public Member Functions

- static sl::String getSDKVersion ()
  Returns the version of the currently installed ZED SDK.
- static int isZEDconnected ()
  Checks if ZED cameras are connected, can be called before instantiating a Camera object.
- static sl::ERROR_CODE sticktoCPUCore (int cpu_core)
  ONLY FOR NVIDIA JETSON : Sticks the calling thread to a specific CPU core.

Detailed Description

The main class to use the ZED camera.

Constructor & Destructor Documentation

Camera()

```
Camera ( )
```

Default constructor which creates an empty Camera.

~Camera()

```
~Camera ( )
```

Camera destructor.

Member Function Documentation

open()

```
ERROR_CODE open (
            InitParameters init_parameters = InitParameters() )
```

Opens the ZED camera in the desired mode (live/SVO), sets all the defined parameters, checks hardware requirements and launch internal self calibration.

Parameters

init_parameters	: a structure containing all the individual parameters. default : a preset of InitParameters.

Returns

An error code given informations about the internal process, if sl::ERROR_CODE::SUCCESS is returned, the camera is ready to use. Every other code indicates an error and the program should be stopped.

isOpened()

```
bool isOpened ( )  [inline]
```

Tests if the camera is opened and running.

Returns

true if the ZED is already setup, otherwise false.

close()

```
void close ( )
```

Closes the camera and free the memory. Camera::open can then be called again to reset the camera if needed.

grab()

```
ERROR_CODE grab (
            RuntimeParameters rt_parameters = RuntimeParameters() )
```

Grabs a new image, rectifies it and computes the depth map. This function is typically called in the main loop.

Parameters

rt_parameters	: a structure containing all the individual parameters. default : a preset of RuntimeParameters.

Returns

An sl::ERROR_CODE::SUCCESS if no problem was encountered, sl::ERROR_CODE::ERROR_COD↩ E_NOT_A_NEW_FRAME otherwise if something wrong happens

retrieveImage()

```
ERROR_CODE retrieveImage (
            Mat & mat,
            VIEW view = VIEW_LEFT,
            MEM type = MEM_CPU,
            int width = 0,
            int height = 0 )
```

Retrieves the desired image (Left, colorized_depth ...).

The retrieve function should be called after the function sl::Camera::grab

Parameters

mat	: [out] the sl::Mat to store the image.
view	: defines the image you want (see sl::VIEW). default : VIEW_LEFT.
type	: the type of the memory of provided mat that should by used. default : MEM_CPU.
width	: if specified, define the width of the output mat. If set to 0, the width of the ZED resolution will be taken. default : 0.
height	: if specified, define the height of the output mat. If set to 0, the height of the ZED resolution will be taken. default : 0.

Returns

SUCCESS if the method succeeded, ERROR_CODE_FAILURE if an error occurred.

getResolution()

```
Resolution getResolution ( )
```

Returns the current image size.

Returns

The image resolution.

getCUDAContext()

```
CUcontext getCUDAContext ( )
```

Returns the CUDA context used for all the computation.

Returns

The CUDA context created by the inner process.

getCameraInformation()

```
CameraInformation getCameraInformation (
            Resolution resizer = Resolution(0, 0) )
```

Returns camera informations (calibration parameters, serial number and current firmware version).

Parameters

	You can specifiy a Resolution different from default image resolution to get the scaled camera informations. default = (0,0) meaning original image size.

Returns

CameraInformation containing the calibration parameters of the ZED, as well as serial number and firmware version It also returns the ZED Serial Number (as uint) (Live or SVO) and the ZED Firmware version (as uint), 0 if the ZED is not connected.

getCameraSettings()

```
int getCameraSettings (
            CAMERA_SETTINGS setting )
```

Returns the current value to the corresponding sl::CAMERA_SETTINGS (Gain, brightness, hue, exposure...).

Parameters

setting	: enum for the control mode.

Returns

The current value for the corresponding control (-1 if something wrong happened).

Note

Works only if the camera is open in live mode.

setCameraSettings()

```
void setCameraSettings (
            CAMERA_SETTINGS settings,
            int value,
            bool use_default = false )
```

Sets the value to the corresponding sl::CAMERA_SETTINGS (Gain, brightness, hue, exposure...).

Parameters

settings	: enum for the control mode.
value	: value to set for the corresponding control.
use_default	: will set default (or automatic) value if set to true (value (int) will not be taken into account). default : false.

Note

Works only if the camera is open in live mode.

getCameraFPS()

```
float getCameraFPS ( )
```

Returns the current FPS of the camera.

Returns

The current FPS (or recorded FPS for SVO). Return -1.f if something goes wrong.

setCameraFPS()

```
void setCameraFPS (
            int desired_fps )
```

Sets a new frame rate for the camera, or the closest available frame rate.

Parameters

desired_fps	: the new desired frame rate.

Note

Works only if the camera is open in live mode.

getCurrentFPS()

```
float getCurrentFPS ( )
```

Returns the current FPS of the application/callback.
It is based on the difference of camera timestamps between two successful grab().

Returns

The current FPS of the application (if grab leads the application) or callback (if ZED is called in a thread)

getCameraTimestamp()

```
timeStamp getCameraTimestamp ( )
```

Returns the timestamp at the time the frame has been extracted from USB stream. (should be called after a grab()).

Returns

The timestamp of the frame grab in ns. -1 if not available (SVO file without compression).

Note

SVO file from SDK 1.0.0 (with compression) contains the camera timestamp for each frame.

getCurrentTimestamp()

```
timeStamp getCurrentTimestamp ( )
```

Returns the current timestamp at the time the function is called. Can be compared to the camera get↩ CameraTimestamp for synchronization.
Use this function to compare the current timestamp and the camera timestamp, since they have the same reference (Computer start time).

Returns

The current timestamp in ns.

getFrameDroppedCount()

```
unsigned int getFrameDroppedCount ( )
```

Returns the number of frame dropped since sl::Camera::grab has been called for the first time.
Based on camera timestamp and FPS comparison.

Returns

The number of frame dropped since first sl::Camera::grab call.

getSVOPosition()

```
int getSVOPosition ( )
```

Returns the current position of the SVO file.

Returns

The current position in the SVO file as int (-1 if the SDK is not reading a SVO).

Note

Works only if the camera is open in SVO reading mode.

setSVOPosition()

```
void setSVOPosition (
            int frame_number )
```

Sets the position of the SVO file to a desired frame.

Parameters

frame_number	: the number of the desired frame to be decoded.

Note

Works only if the camera is open in SVO playback mode.

getSVONumberOfFrames()

```
int getSVONumberOfFrames ( )
```

Returns the number of frames in the SVO file.

Returns

The total number of frames in the SVO file (-1 if the SDK is not reading a SVO).

Note

Works only if the camera is open in SVO reading mode.

retrieveMeasure()

```
ERROR_CODE retrieveMeasure (
            Mat & mat,
            MEASURE measure = MEASURE_DEPTH,
            MEM type = MEM_CPU,
            int width = 0,
            int height = 0 )
```

Retrieves the desired measure (disparity, depth, point cloud ...).

The retrieve function should be called after the function sl::Camera::grab

Parameters

mat	: [out] the sl::Mat to store the measures.
measure	: defines the measure you want. (see sl::MEASURE), default : MEASURE_DEPTH
type	: the type of the memory of provided mat that should by used. default : MEM_CPU.
width	: if specified, define the width of the output mat. If set to 0, the width of the ZED resolution will be taken. default : 0
height	: if specified, define the height of the output mat. If set to 0, the height of the ZED resolution will be taken. default : 0

Returns

SUCCESS if the method succeeded, ERROR_CODE_FAILURE if an error occurred.

getDepthMaxRangeValue()

```
float getDepthMaxRangeValue ( )
```

Returns the current maximum distance of depth/disparity estimation.

Returns

The current maximum distance that can be computed in the defined sl::UNIT.

setDepthMaxRangeValue()

```
void setDepthMaxRangeValue (
            float depth_max_range )
```

Sets the maximum distance of depth/disparity estimation (all values after this limit will be reported as sl↩ ::TOO_FAR value).

Parameters

depth_max_range	: maximum distance in the defined sl::UNIT.

getDepthMinRangeValue()

```
float getDepthMinRangeValue ( )
```

Returns the closest measurable distance by the camera, according to the camera and the depth map parameters.

Returns

The minimum distance that can be computed in the defined sl::UNIT.

getConfidenceThreshold()

```
int getConfidenceThreshold ( )
```

Returns the current confidence threshold value apply to the disparity map(and by extension the depth map).

Returns

The current threshold value between 0 and 100.

setConfidenceThreshold()

```
void setConfidenceThreshold (
            int conf_threshold_value )
```

Sets a threshold for the disparity map confidence (and by extension the depth map).

A lower value means more confidence and precision (but less density), an upper value reduces the filtering (more density, less certainty).

Parameters

conf_threshold_value	: a value in [1,100].

enableTracking()

```
ERROR_CODE enableTracking (
            TrackingParameters tracking_parameters = TrackingParameters() )
```

Initializes and start the tracking processes.

Parameters

tracking_parameters	: Structure of sl::TrackingParameters, which defines specific parameters for tracking. default : a preset of sl::TrackingParameters.

Returns

sl::ERROR_CODE_FAILURE if the sl::TrackingParameters::area_file_path file wasn't found, sl::SUC↩ CESS otherwise.

Warning

The area localization is a beta feature, the behavior might change in the future.

getPosition()

```
sl::TRACKING_STATE getPosition (
            sl::Pose & camera_pose,
            REFERENCE_FRAME reference_frame = sl::REFERENCE_FRAME_WORLD )
```

Fills the position of the camera frame in the world frame and return the current state of the Tracker.

Note

The camera frame is positioned at the back of the left eye of the ZED.

Parameters

camera_pose	[out] : the pose containing the position of the camera (path or position) and other information (timestamp, confidence)
reference_frame	: defines the reference from which you want the pose to be expressed. default : sl::REFERENCE_FRAME_WORLD.

Returns

The current state of the tracking process.

Extract Rotation Matrix : camera_pose.getRotation();
Extract Translation Vector: camera_pose.getTranslation();
Convert to Orientation / quaternion : camera_pose.getOrientation();

getAreaExportState()

```
sl::AREA_EXPORT_STATE getAreaExportState ( )
```

Returns the state of the spatial memory export process.

Returns

The current state of the spatial memory export process

disableTracking()

```
void disableTracking (
            sl::String area_file_path = "" )
```

Disables motion tracking.

Parameters

area_file_path	: if set, save the spatial memory database in a '.area' file. default : (empty) areaFilePath is the name and path of the database, e.g. : "path/to/file/myArea1.area".

Warning

This feature is still in beta, you might encounter reloading issues.
Please also note that the '.area' database depends on the depth map sl::SENSING_MODE chosen during the recording. The same mode must be used to reload the database.

Note

The saving is done asynchronously, the state can be get by getAreaExportState().

resetTracking()

```
ERROR_CODE resetTracking (
            sl::Transform & path )
```

Resets the tracking, re-initializes the path with the transformation matrix given.

Note

Please note that this function will also flush the area database built / loaded.

enableSpatialMapping()

```
ERROR_CODE enableSpatialMapping (
            SpatialMappingParameters spatial_mapping_parameters = SpatialMappingParameters()
)
```

Initializes and starts the spatial mapping processes.

The spatial mapping will create a geometric representation of the scene based on both tracking data and 3D point clouds.
The resulting output is a sl::Mesh and can be obtained by the sl::Camera::extractWholeMesh function or with sl::Camera::retrieveMeshAsync after calling sl::Camera::requestMeshAsync.

Parameters

spatial_mapping_parameters	: the structure containing all the specific parameters for the spatial mapping. default : a balanced parameters preset between geometric fidelity and output file size. For more informations, checkout the sl::SpatialMappingParameters documentation.

Returns

sl::SUCCESS if everything went fine, sl::ERROR_CODE_FAILURE otherwise

Warning

The tracking needs to be enabled to create a map.
The performance greatly depends on the input parameters. If the mapping framerate is too slow in live mode, consider using a SVO file, or choose a coarser mesh resolution

Note

This features is using host memory (RAM) to store the 3D map, the maximum amount of available memory allowed can be tweaked using the SpatialMappingParameters.

getSpatialMappingState()

```
SPATIAL_MAPPING_STATE getSpatialMappingState ( )
```

Returns the current spatial mapping state.

Returns

The current state of the spatial mapping process

requestMeshAsync()

```
void requestMeshAsync ( )
```

Starts the mesh generation process in a non blocking thread from the spatial mapping process.

Note

Only one mesh generation can be done at a time, consequently while the previous launch is not done every call will be ignored.

getMeshRequestStatusAsync()

```
ERROR_CODE getMeshRequestStatusAsync ( )
```

Returns the mesh generation status, useful after calling requestMeshAsync to know if you can call sl::↩ Camera::retrieveMeshAsync().

Returns

sl::SUCCESS if the mesh is ready and not yet retrieved, otherwise sl::ERROR_CODE_FAILURE.

retrieveMeshAsync()

```
ERROR_CODE retrieveMeshAsync (
            sl::Mesh & mesh )
```

Retrieves the generated mesh after calling requestMeshAsync.

Parameters

mesh	[out] : The mesh to be filled.

Returns

sl::SUCCESS if the mesh is retrieved, otherwise sl::ERROR_CODE_FAILURE.

extractWholeMesh()

```
ERROR_CODE extractWholeMesh (
            sl::Mesh & mesh )
```

Extracts the current mesh from the spatial mapping process.

Note

This function will return when the mesh has been created or updated. This is therefore a blocking function. You should either call it in a thread or at the end of the mapping process. Calling this function in the grab loop will block the depth and tracking computation and therefore gives bad results.

Parameters

mesh	[out] : The mesh to be filled.

Returns

sl::SUCCESS if the mesh is filled and available, otherwise sl::ERROR_CODE_FAILURE.

pauseSpatialMapping()

```
void pauseSpatialMapping (
            bool status )
```

Switches the pause status of the data integration mechanism for the spatial mapping.

Parameters

status	: if true, the integration is paused. If false, the spatial mapping is resumed.

disableSpatialMapping()

```
void disableSpatialMapping ( )
```

Disables the Spatial Mapping process. All the spatial mapping functions are disables, mesh cannot be retrieves after this call.

getSelfCalibrationState()

```
SELF_CALIBRATION_STATE getSelfCalibrationState ( )
```

Returns the current status of the self-calibration.

Returns

A status code given informations about the self calibration status.

For more details see sl::SELF_CALIBRATION_STATE.

resetSelfCalibration()

```
void resetSelfCalibration ( )
```

Resets the self camera calibration. This function can be called at any time AFTER the sl::Camera::open function has been called.

It will reset and calculate again correction for misalignment, convergence and color mismatch. It can be called after changing camera parameters without needing to restart your executable.

If no problem was encountered, the camera will use new parameters. Otherwise, it will be the old ones.

enableRecording()

```
ERROR_CODE enableRecording (
            sl::String video_filename,
            SVO_COMPRESSION_MODE compression_mode = SVO_COMPRESSION_MODE_LOSSLESS )
```

Creates a file for recording the current frames.

Parameters

video_filename	: can be a ∗.svo file or a ∗.avi file (detected by the suffix name provided).
compression_mode	: can be one of the sl::SVO_COMPRESSION_MODE enum. defaylt : SVO_COMPRESSION_MODE_LOSSLESS.

Warning

This function can be called multiple times during ZED lifetime, but if video_filename is already existing, the file will be erased.

Returns

an sl::ERROR_CODE that defines if file was successfully created and can be filled with images.

record()

```
sl::RecordingState record ( )
```

Records the current frame provided by grab() into the file.

Warning

grab() must be called before record() to take the last frame available. Otherwise, it will be the last grabbed frame.

Returns

The recording state structure, for more details see sl::RecordingState.

disableRecording()

```
void disableRecording ( )
```

Disables the recording and closes the generated file.

getSDKVersion()

```
static sl::String getSDKVersion ( )  [static]
```

Returns the version of the currently installed ZED SDK.

Returns

The ZED SDK version as a string with the following format : MAJOR.MINOR.PATCH

isZEDconnected()

```
static int isZEDconnected ( )  [static]
```

Checks if ZED cameras are connected, can be called before instantiating a Camera object.

Returns

The number of connected ZED.

Warning

On Windows, only one ZED is accessible so this function will return 1 even if multiple ZED are connected.

sticktoCPUCore()

```
static sl::ERROR_CODE sticktoCPUCore (
            int cpu_core )  [static]
```

ONLY FOR NVIDIA JETSON : Sticks the calling thread to a specific CPU core.

Parameters

cpuCore	: int that defines the core the thread must be run on. could be between 0 and 3. (cpu0,cpu1,cpu2,cpu3).

Returns

sl::SUCCESS if stick is OK, otherwise status error.

Warning

Function only available for Nvidia Jetson. On other platform, result will be always 0 and no operations are performed.

CameraInformation Struct Reference

Camera specific parameters.

Public Attributes

- CalibrationParameters calibration_parameters
- CalibrationParameters calibration_parameters_raw
- unsigned int serial_number = 0
- unsigned int firmware_version = 0

Detailed Description

Camera specific parameters.

Member Data Documentation

calibration_parameters

```
CalibrationParameters calibration_parameters
```

Intrinsic and Extrinsic stereo parameters for rectified images (default).

calibration_parameters_raw

```
CalibrationParameters calibration_parameters_raw
```

Intrinsic and Extrinsic stereo parameters for original images (unrectified).

serial_number

```
unsigned int serial_number = 0
```

camera dependent serial number.

firmware_version

```
unsigned int firmware_version = 0
```

current firmware version of the camera.

CameraParameters Struct Reference

Intrinsic parameters of a camera.

Public Member Functions

- void SetUp (float focal_x, float focal_y, float center_x, float center_y)
  Setups the parameter of a camera.

Public Attributes

- float fx
- float fy
- float cx
- float cy
- double disto [5]
- float v_fov
- float h_fov
- float d_fov
- Resolution image_size

Detailed Description

Intrinsic parameters of a camera.

Note

Similar to the CalibrationParameters, those parameters are taken from the settings file (SNXXX.conf) and are modified during the sl::Camera::open call (with or without Self-Calibration). Those parameters given after sl::Camera::open call, represent the "new camera matrix" that fits/defines each image taken after rectification (through retrieveImage).
fx,fy,cx,cy must be the same for Left and Right Camera once sl::Camera::open has been called. Since distortion is corrected during rectification, distortion should not be considered after sl::Camera::open call.

Member Function Documentation

SetUp()

```
void SetUp (
            float focal_x,
            float focal_y,
            float center_x,
            float center_y )  [inline]
```

Setups the parameter of a camera.

size in pixels of the images given by the camera.

Parameters

focal_x	: horizontal focal length.

Parameters

focal↩ _y	: vertical focal length.
focal↩ _x	: horizontal optical center.
focal↩ _x	: vertical optical center.

Member Data Documentation

fx

```
float fx
```

Focal length in pixels along x axis.

fy

```
float fy
```

Focal length in pixels along y axis.

cx

```
float cx
```

Optical center along x axis, defined in pixels (usually close to width/2).

cy

```
float cy
```

Optical center along y axis, defined in pixels (usually close to height/2).

disto

```
double disto[5]
```

Distortion factor : [k1, k2, p1, p2, k3]. Radial (k1,k2,k3) and Tangential (p1,p2) distortion.

v_fov

```
float v_fov
```

Vertical field of view after stereo rectification, in degrees.

h_fov

```
float h_fov
```

Horizontal field of view after stereo rectification, in degrees.

d_fov

```
float d_fov
```

Diagonal field of view after stereo rectification, in degrees.

image_size

```
Resolution image_size
```

Chunk Class Reference

Represents a sub mesh, it contains local vertices and triangles.

Public Member Functions

- Chunk ()

  Default constructor which creates an empty sl::Chunk.
- ~Chunk ()

  sl::Chunk destructor.
- void clear ()

  Clear all Chunk data.

Public Attributes

- std::vector< sl::float3 > vertices
- std::vector< sl::uint3 > triangles
- std::vector< sl::float3 > normals
- std::vector< sl::float2 > uv
- unsigned long long timestamp
- sl::float3 barycenter
- bool has_been_updated

Detailed Description

Represents a sub mesh, it contains local vertices and triangles.

Vertices and normals have the same size and are linked by id stored in triangles.

Note

uv contains data only if your mesh have textures (by loading it or after calling sl::Mesh::applyTexture).

Constructor & Destructor Documentation

Chunk()

```
Chunk ( )
```

Default constructor which creates an empty sl::Chunk.

~Chunk()

```
~Chunk ( )
```

sl::Chunk destructor.

Member Function Documentation

clear()

```
void clear ( )
```

Clear all Chunk data.

Member Data Documentation

vertices

```
std::vector<sl::float3> vertices
```

Vertices are defined by a 3D point {x,y,z}.

triangles

```
std::vector<sl::uint3> triangles
```

Triangles (or faces) contains the index of its three vertices. It corresponds to the 3 vertices of the triangle {v1, v2, v3}.

normals

```
std::vector<sl::float3> normals
```

Normals are defined by three components, {nx, ny, nz}. Normals are defined for each vertices.

uv

```
std::vector<sl::float2> uv
```

Texture coordinates defines 2D points on a texture.

Note

Contains data only if your mesh have textures (by loading it or calling applytexture).

timestamp

```
unsigned long long timestamp
```

Timestamp of the latest update.

barycenter

```
sl::float3 barycenter
```

3D centroid of the Chunk.

has_been_updated

```
bool has_been_updated
```

true if the chunk has been updated by an inner process.

InitParameters Class Reference

Parameters that will be fixed for the whole execution life time of the sl::Camera.

Public Member Functions

- InitParameters (RESOLUTION camera_resolution_=RESOLUTION_HD720, int camera_fps_=0, int camera_linux_id_=0, sl::String svo_input_filename_=sl::String(), bool svo_real_time_mode_=false, DEPTH_MODE depth_mode_=DEPTH_MODE_PERFORMANCE, UNIT coordinate_units_=UNIT_MILLIMETER, COORDINATE_SYSTEM coordinate_system_=COORDINATE_SYSTEM_IMAGE, bool sdk_verbose_=false, int sdk_gpu_id_=-1, float depth_minimum_distance_=-1., bool camera_disable_self_calib_=false, bool camera_image_flip_=false, bool enable_right_side_measure_=false, int camera_buffer_count_linux_=4, sl::String sdk_verbose_log_file_=sl::String(), bool depth_stabilization_=true)

  Default constructor, set all parameters to their default and optimized values.
- bool save (sl::String filename)

  Saves the current bunch of parameters into a file.
- bool load (sl::String filename)

  Loads the values of the parameters contained in a file.

Public Attributes

- RESOLUTION camera_resolution
- int camera_fps
- int camera_image_flip
- bool camera_disable_self_calib
- bool enable_right_side_measure
- int camera_buffer_count_linux
- int camera_linux_id
- sl::String svo_input_filename
- bool svo_real_time_mode
- DEPTH_MODE depth_mode
- bool depth_stabilization
- float depth_minimum_distance
- UNIT coordinate_units
- COORDINATE_SYSTEM coordinate_system
- int sdk_gpu_id
- bool sdk_verbose
- sl::String sdk_verbose_log_file

Detailed Description

Parameters that will be fixed for the whole execution life time of the sl::Camera.

By default it open the ZED camera in live mode at RESOLUTION_HD720 and set the depth mode to DEPTH_MODE_PERFORMANCE to get low computation time.
You can customize it to fit your application and then save it to create a preset that can be loaded for further executions.

Constructor & Destructor Documentation

InitParameters()

```
InitParameters (
            RESOLUTION camera_resolution_ = RESOLUTION_HD720,
            int camera_fps_ = 0,
            int camera_linux_id_ = 0,
```

```
            sl::String svo_input_filename_ = sl::String(),
            bool svo_real_time_mode_ = false,
            DEPTH_MODE depth_mode_ = DEPTH_MODE_PERFORMANCE,
            UNIT coordinate_units_ = UNIT_MILLIMETER,
            COORDINATE_SYSTEM coordinate_system_ = COORDINATE_SYSTEM_IMAGE,
            bool sdk_verbose_ = false,
            int sdk_gpu_id_ = -1,
            float depth_minimum_distance_ = -1.,
            bool camera_disable_self_calib_ = false,
            bool camera_image_flip_ = false,
            bool enable_right_side_measure_ = false,
            int camera_buffer_count_linux_ = 4,
            sl::String sdk_verbose_log_file_ = sl::String(),
            bool depth_stabilization_ = true )  [inline]
```

Default constructor, set all parameters to their default and optimized values.

Member Function Documentation

save()

```
bool save (
            sl::String filename )
```

Saves the current bunch of parameters into a file.

Parameters

filename	: the path to the file in which the parameters will be stored.

Returns

true if file was successfully saved, otherwise false.

load()

```
bool load (
            sl::String filename )
```

Loads the values of the parameters contained in a file.

Parameters

filename	: the path to the file from which the parameters will be loaded.

Returns

true if the file was successfully loaded, otherwise false.

Member Data Documentation

camera_resolution

```
RESOLUTION camera_resolution
```

Define the chosen ZED resolution
default : RESOLUTION_HD720

camera_fps

```
int camera_fps
```

Requested FPS for this resolution. set as 0 will choose the default FPS for this resolution (see User guide).
default : 0

camera_image_flip

```
int camera_image_flip
```

Defines if the image are horizontally flipped.
default : false

camera_disable_self_calib

```
bool camera_disable_self_calib
```

If set to true, it will disable self-calibration and take the optional calibration parameters without optimizing them.
It is advised to leave it as false, so that calibration parameters can be optimized.
default : false

enable_right_side_measure

```
bool enable_right_side_measure
```

Defines if right MEASURE should be computed (needed for MEASURE_<XXX>_RIGHT)
default : false

camera_buffer_count_linux

```
int camera_buffer_count_linux
```

ONLY for LINUX : Set the number of buffers in the internal grabbing process.
Decrease this number may reduce latency but can also produce more corrupted frames.
default: 4

Warning

Linux Only, this parameter has no effect on Windows.

camera_linux_id

```
int camera_linux_id
```

ONLY for LINUX : if multiple ZEDs are connected, it will choose the first zed listed (if zed_linux_id=0), the second listed (if zed_linux_id=1), ...
Each ZED will create its own memory (CPU and GPU), therefore the number of ZED available will depend on the configuration of your computer.
default : 0

Warning

Linux Only, this parameter has no effect on Windows.

svo_input_filename

```
sl::String svo_input_filename
```

Path with filename to the recorded SVO file.
default : (empty)

svo_real_time_mode

```
bool svo_real_time_mode
```

When enabled the timestamp is taken as reference to determine the reading framerate.
This mode simulates the live camera and consequently skipped frames if the computation framerate is too slow.
default : false

depth_mode

```
DEPTH_MODE depth_mode
```

Defines the quality of the depth map, affects the level of details and also the computation time.
default : sl::DEPTH_MODE::DEPTH_MODE_PERFORMANCE

depth_stabilization

```
bool depth_stabilization
```

Defines if the depth map should be stabilize.
This requires the positional tracking data, it will be enabled automatically if needed.
default : true

depth_minimum_distance

```
float depth_minimum_distance
```

Specify the minimum depth value that will be computed, in the sl::UNIT you define.
In case of limited computation power, consider increasing the value.
default : 70cm (-1)

Warning

> The computation time is affected by the value. The closer it gets the longer the sl::Camera::grab() will take.

coordinate_units

```
UNIT coordinate_units
```

Define the unit for all the metric values (depth, point cloud, tracking, mesh).
default : sl::UNIT::UNIT_MILLIMETER

coordinate_system

```
COORDINATE_SYSTEM coordinate_system
```

Define the coordinate system of the world frame (and the camera frame as well).
This defines the order and the direction of the axis of the coordinate system. see COORDINATE_SYSTEM for more information.
default : sl::COORDINATE_SYSTEM::COORDINATE_SYSTEM_IMAGE

sdk_gpu_id

```
int sdk_gpu_id
```

Defines the graphics card on which the computation will be done. The default value search the more powerful (most CUDA cores) usable GPU.
default : -1

sdk_verbose

```
bool sdk_verbose
```

Defines if you want the SDK provides text feedback in the console. If set to true, it will output some information about the current status of initialization.
default : false

sdk_verbose_log_file

```
sl::String sdk_verbose_log_file
```

Store the program outputs into the log file defined by its filename.
default : (empty)

Note

it will redirect std::cout calls for the program using the ZED SDK (including SDK verbosity outputs) in the log file.
Can be used with Unreal, Unity or any software that doesn't have a standard console output.

Mat Class Reference

The Mat class can handle multiple matrix format from 1 to 4 channels, with different value types (float or uchar), and can be stored CPU and/or GPU side.

Public Member Functions

- Mat ()

  empty Mat default constructor.
- Mat (size_t width, size_t height, MAT_TYPE mat_type, MEM memory_type=MEM_CPU)

  Mat constructor.
- Mat (size_t width, size_t height, MAT_TYPE mat_type, sl::uchar1 *ptr, size_t step, MEM memory_type=MEM_CPU)

  Mat constructor from an existing data pointer.
- Mat (size_t width, size_t height, MAT_TYPE mat_type, sl::uchar1 *ptr_cpu, size_t step_cpu, sl::uchar1 *ptr_gpu, size_t step_gpu)

  Mat constructor from two existing data pointers, CPU and GPU.
- Mat (sl::Resolution resolution, MAT_TYPE mat_type, MEM memory_type=MEM_CPU)

  Mat constructor.
- Mat (sl::Resolution resolution, MAT_TYPE mat_type, sl::uchar1 *ptr, size_t step, MEM memory_type=MEM_CPU)

  Mat constructor from an existing data pointer.
- Mat (sl::Resolution resolution, MAT_TYPE mat_type, sl::uchar1 *ptr_cpu, size_t step_cpu, sl::uchar1 *ptr_gpu, size_t step_gpu)

  Mat constructor from two existing data pointers, CPU and GPU.
- Mat (const sl::Mat &mat)

  Mat constructor by copy (deep copy).
- void alloc (size_t width, size_t height, MAT_TYPE mat_type, MEM memory_type=MEM_CPU)

  Allocates the Mat memory.
- void alloc (sl::Resolution resolution, MAT_TYPE mat_type, MEM memory_type=MEM_CPU)

  Allocates the Mat memory.
- ~Mat ()

Mat destructor. This function calls Mat::free to release owned memory.

- void free (MEM memory_type=MEM_CPU|MEM_GPU)

  Free the owned memory.
- Mat & operator= (const Mat &that)

  Performs a shallow copy.
  This function doesn't copy the data array, it only copies the pointer.
- ERROR_CODE updateCPUfromGPU ()

  Downloads data from DEVICE (GPU) to HOST (CPU), if possible.
- ERROR_CODE updateGPUfromCPU ()

  Uploads data from HOST (CPU) to DEVICE (GPU), if possible.
- ERROR_CODE copyTo (Mat &dst, COPY_TYPE cpyType=COPY_TYPE_CPU_CPU) const

  Copies data an other Mat (deep copy).
- ERROR_CODE setFrom (const Mat &src, COPY_TYPE cpyType=COPY_TYPE_CPU_CPU)

  Copies data from an other Mat (deep copy).
- ERROR_CODE read (const char *filePath)

  Reads an image from a file (only if sl::MEM_CPU is available on the current sl::Mat).
- ERROR_CODE write (const char *filePath)

  Writes the sl::Mat (only if sl::MEM_CPU is available) into a file as an image.
- template<typename T >
  ERROR_CODE setTo (T value, MEM memory_type=MEM_CPU)

  Fills the Mat with the given value.
- template<typename N >
  ERROR_CODE setValue (size_t x, size_t y, N value, MEM memory_type=MEM_CPU)

  Sets a value to a specific point in the matrix.
- template<typename N >
  ERROR_CODE getValue (size_t x, size_t y, N *value, MEM memory_type=MEM_CPU) const

  Returns the value of a specific point in the matrix.
- size_t getWidth () const
- size_t getHeight () const
- Resolution getResolution () const
- size_t getChannels () const
- MAT_TYPE getDataType () const
- MEM getMemoryType () const
- template<typename N >
  N * getPtr (MEM memory_type=MEM_CPU) const
- size_t getStepBytes (MEM memory_type=MEM_CPU) const
- template<typename N >
  size_t getStep (MEM memory_type=MEM_CPU) const
- size_t getStep (MEM memory_type=MEM_CPU) const
- size_t getPixelBytes () const
- size_t getWidthBytes () const
- sl::String getInfos ()
- bool isInit () const
- bool isMemoryOwner () const
- ERROR_CODE clone (const Mat &src)

  Duplicates Mat by copy (deep copy).
- ERROR_CODE move (Mat &dst)

  Moves Mat data to another Mat.

Static Public Member Functions

- static void swap (sl::Mat &mat1, sl::Mat &mat2)

  Swaps the content of the provided Mat (only swaps the pointers, no data copy).

Public Attributes

- sl::String name
- bool verbose = false

Detailed Description

The Mat class can handle multiple matrix format from 1 to 4 channels, with different value types (float or uchar), and can be stored CPU and/or GPU side.

sl::Mat is defined in a row-major order, it means that, for an image buffer, the entire first row is stored first, followed by the entire second row, and so on.

The CPU and GPU buffer aren't automatically synchronized for performance reasons, you can use Mat↩ ::updateCPUfromGPU / Mat::updateGPUfromCPU to do it. If you are using the GPU side of the Mat object, you need to make sure to call sl::Mat::free() before destroying the sl::Camera object. The destruction of the sl::Camera object delete the CUDA context needed to free the GPU Mat memory.

Constructor & Destructor Documentation

Mat() [1/8]

```
Mat ( )
```

empty Mat default constructor.

Mat() [2/8]

```
Mat (
            size_t width,
            size_t height,
            MAT_TYPE mat_type,
            MEM memory_type = MEM_CPU )
```

Mat constructor.

Parameters

width	: width of the matrix in pixels.
height	: height of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
memory_type	: defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU). This function directly allocates the requested memory. It calls Mat::alloc.

Mat() [3/8]

```
Mat (
            size_t width,
            size_t height,
            MAT_TYPE mat_type,
            sl::uchar1 * ptr,
            size_t step,
            MEM memory_type = MEM_CPU )
```

Mat constructor from an existing data pointer.

Parameters

width	: width of the matrix in pixels.
height	: height of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
ptr	: pointer to the data array. (CPU or GPU).
step	: step of the data array. (the Bytes size of one pixel row)
memory_type	: defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU). This function doesn't allocate the memory.

Mat() [4/8]

```
Mat (
            size_t width,
            size_t height,
            MAT_TYPE mat_type,
            sl::uchar1 * ptr_cpu,
            size_t step_cpu,
            sl::uchar1 * ptr_gpu,
            size_t step_gpu )
```

Mat constructor from two existing data pointers, CPU and GPU.

Parameters

width	: width of the matrix in pixels.
height	: height of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
ptr_cpu	: CPU pointer to the data array.
step_cpu	: step of the CPU data array. (the Bytes size of one pixel row)
ptr_gpu	: GPU pointer to the data array.
step_gpu	: step of the GPU data array. (the Bytes size of one pixel row) This function doesn't allocate the memory.

Mat() [5/8]

```
Mat (
            sl::Resolution resolution,
            MAT_TYPE mat_type,
            MEM memory_type = MEM_CPU )
```

Mat constructor.

Parameters

resolution	: the size of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
memory_type	: defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU). This function directly allocates the requested memory. It calls Mat::alloc.

Mat() [6/8]

```
Mat (
            sl::Resolution resolution,
            MAT_TYPE mat_type,
            sl::uchar1 * ptr,
            size_t step,
            MEM memory_type = MEM_CPU )
```

Mat constructor from an existing data pointer.

Parameters

resolution	: the size of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
ptr	: pointer to the data array. (CPU or GPU).
step	: step of the data array. (the Bytes size of one pixel row)
memory_type	: defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU). This function doesn't allocate the memory.

Mat() [7/8]

```
Mat (
            sl::Resolution resolution,
            MAT_TYPE mat_type,
            sl::uchar1 * ptr_cpu,
            size_t step_cpu,
            sl::uchar1 * ptr_gpu,
            size_t step_gpu )
```

Mat constructor from two existing data pointers, CPU and GPU.

Parameters

resolution	: the size of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
ptr_cpu	: CPU pointer to the data array.
step_cpu	: step of the CPU data array. (the Bytes size of one pixel row)
ptr_gpu	: GPU pointer to the data array.
step_gpu	: step of the GPU data array. (the Bytes size of one pixel row) This function doesn't allocate the memory.

Mat() [8/8]

```
Mat (
            const sl::Mat & mat )
```

Mat constructor by copy (deep copy).

Parameters

mat	: the reference to the sl::Mat to copy. This function allocates and duplicates the data

~Mat()

```
~Mat ( )
```

Mat destructor. This function calls Mat::free to release owned memory.

Member Function Documentation

alloc() [1/2]

```
void alloc (
             size_t width,
             size_t height,
             MAT_TYPE mat_type,
             MEM memory_type = MEM_CPU )
```

Allocates the Mat memory.

Parameters

width	: width of the matrix in pixels
height	: height of the matrix in pixels
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
memory_type	: defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU).

Warning

It erases previously allocated memory.

alloc() [2/2]

```
void alloc (
             sl::Resolution resolution,
             MAT_TYPE mat_type,
             MEM memory_type = MEM_CPU )
```

Allocates the Mat memory.

Parameters

resolution	: the size of the matrix in pixels.
mat_type	: the type of the matrix (sl::MAT_TYPE_32F_C1,sl::MAT_TYPE_8U_C4...)
memory_type	: defines where the buffer will be stored. (sl::MEM_CPU and/or sl::MEM_GPU).

Warning

It erases previously allocated memory.

free()

```
void free (
             MEM memory_type = MEM_CPU|MEM_GPU )
```

Free the owned memory.

Parameters

memory_type	: specify whether you want to free the sl::MEM_CPU and/or sl::MEM_GPU memory.

operator=()

```
Mat& operator= (
            const Mat & that )
```

Performs a shallow copy.
This function doesn't copy the data array, it only copies the pointer.

Parameters

that	: the sl::Mat to be copied.

Returns

The new sl::Mat object which point to the same data as that.

updateCPUfromGPU()

```
ERROR_CODE updateCPUfromGPU ( )
```

Downloads data from DEVICE (GPU) to HOST (CPU), if possible.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Note

If no CPU or GPU memory are available for this Mat, some are directly allocated.
If verbose sets, you have informations in case of failure.

updateGPUfromCPU()

```
ERROR_CODE updateGPUfromCPU ( )
```

Uploads data from HOST (CPU) to DEVICE (GPU), if possible.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Note

If no CPU or GPU memory are available for this Mat, some are directly allocated.
If verbose sets, you have informations in case of failure.

copyTo()

```
ERROR_CODE copyTo (
            Mat & dst,
            COPY_TYPE cpyType = COPY_TYPE_CPU_CPU ) const
```

Copies data an other Mat (deep copy).

Parameters

dst	: the Mat where the data will be copied.
cpyType	: specify the memories that will be used for the copy.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Note

If the destination is not allocated or has a not a compatible sl::MAT_TYPE or sl::Resolution, current memory is freed and new memory is directly allocated.

setFrom()

```
ERROR_CODE setFrom (
            const Mat & src,
            COPY_TYPE cpyType = COPY_TYPE_CPU_CPU )
```

Copies data from an other Mat (deep copy).

Parameters

src	: the Mat where the data will be copied from.
cpyType	: specify the memories that will be used for the update.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Note

If the current Mat is not allocated or has a not a compatible sl::MAT_TYPE or sl::Resolution with the source, current memory is freed and new memory is directly allocated.

read()

```
ERROR_CODE read (
            const char * filePath )
```

Reads an image from a file (only if sl::MEM_CPU is available on the current sl::Mat).

Parameters

filePath	: file path including the name and extension.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Note

Supported sl::MAT_TYPE are : sl::MAT_TYPE_8U_C1, sl::MAT_TYPE_8U_C3 and sl::MAT_TYPE_8↩ U_C4. input files format are PNG and JPEG. verbose sets, you have informations in case of failure.

write()

```
ERROR_CODE write (
            const char * filePath )
```

Writes the sl::Mat (only if sl::MEM_CPU is available) into a file as an image.

Parameters

filePath	: file path including the name and extension.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Note

Supported sl::MAT_TYPE are : sl::MAT_TYPE_8U_C1, sl::MAT_TYPE_8U_C3 and sl::MAT_TYPE_8↩ U_C4. output files format are PNG and JPEG. verbose sets, you have informations in case of failure.

setTo()

```
ERROR_CODE setTo (
            T value,
            MEM memory_type = MEM_CPU )
```

Fills the Mat with the given value.

Parameters

value	: the value to be copied all over the matrix.
memory_type	: defines which buffer to fill, CPU and/or GPU. This function overwrite all the matrix.

Note

This function is templated for sl::uchar1, sl::uchar2, sl::uchar3, sl::uchar4, sl::float1, sl::float2, sl↩ ::float3, sl::float4.

setValue()

```
ERROR_CODE setValue (
            size_t x,
            size_t y,
            N value,
            MEM memory_type = MEM_CPU )
```

Sets a value to a specific point in the matrix.

Parameters

x	: specify the column.

Parameters

y	: specify the row.
value	: the value to be set.
memory_type	: defines which memory will be updated.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Warning

Not efficient for sl::MEM_GPU, use it on sparse data.

Note

This function is templated for sl::uchar1, sl::uchar2, sl::uchar3, sl::uchar4, sl::float1, sl::float2, sl↩ ::float3, sl::float4.

getValue()

```
ERROR_CODE getValue (
             size_t x,
             size_t y,
             N * value,
             MEM memory_type = MEM_CPU ) const
```

Returns the value of a specific point in the matrix.

Parameters

x	: specify the column
y	: specify the row
memory_type	: defines which memory should be read.

Returns

sl::SUCCESS if everything went well, sl::ERROR_CODE_FAILURE otherwise.

Warning

Not efficient for sl::MEM_GPU, use it on sparse data.

Note

This function is templated for sl::uchar1, sl::uchar2, sl::uchar3, sl::uchar4, sl::float1, sl::float2, sl↩ ::float3, sl::float4.

getWidth()

```
size_t getWidth ( ) const  [inline]
```

brief Returns the width of the matrix.

Returns

The width of the matrix in pixels.

getHeight()

```
size_t getHeight ( ) const  [inline]
```

brief Returns the height of the matrix.

Returns

The height of the matrix in pixels.

getResolution()

```
Resolution getResolution ( ) const  [inline]
```

brief Returns the height of the matrix.

Returns

The height of the matrix in pixels.

getChannels()

```
size_t getChannels ( ) const  [inline]
```

brief Returns the number of values stored in one pixel.

Returns

The number of values in a pixel.

getDataType()

```
MAT_TYPE getDataType ( ) const  [inline]
```

brief Returns the format of the matrix.

Returns

The format of the current Mat.

getMemoryType()

```
MEM getMemoryType ( ) const  [inline]
```

brief Returns the type of memory (CPU and/or GPU).

Returns

The type of allocated memory.

getPtr()

```
N* getPtr (
            MEM memory_type = MEM_CPU ) const
```

brief Returns the CPU or GPU data pointer.

Parameters

memory_type	: specify whether you want sl::MEM_CPU or sl::MEM_GPU step.

Returns

The pointer of the Mat data.

getStepBytes()

```
size_t getStepBytes (
            MEM memory_type = MEM_CPU ) const  [inline]
```

brief Returns the memory step in Bytes (the Bytes size of one pixel row).

Parameters

memory_type	: specify whether you want sl::MEM_CPU or sl::MEM_GPU step.

Returns

The step in bytes of the specified memory.

getStep() [1/2]

```
size_t getStep (
            MEM memory_type = MEM_CPU ) const  [inline]
```

brief Returns the memory step in number of elements (the number of values in one pixel row).

Parameters

memory_type	: specify whether you want sl::MEM_CPU or sl::MEM_GPU step.

Returns

The step in number of elements.

getStep() [2/2]

```
size_t getStep (
            MEM memory_type = MEM_CPU ) const  [inline]
```

brief Returns the memory step in number of elements (the number of values in one pixel row).

Parameters

memory_type	: specify whether you want sl::MEM_CPU or sl::MEM_GPU step.

Returns

The step in number of elements.

getPixelBytes()

```
size_t getPixelBytes ( ) const  [inline]
```

brief Returns the size in bytes of one pixel.

Returns

The size in bytes of a pixel.

getWidthBytes()

```
size_t getWidthBytes ( ) const  [inline]
```

brief Returns the size in bytes of a row.

Returns

The size in bytes of a row.

getInfos()

```
sl::String getInfos ( )
```

brief Return the informations about the Mat into a sl::String.

Returns

A string containing the Mat informations.

isInit()

```
bool isInit ( ) const  [inline]
```

brief Defines whether the Mat is initialized or not.

Returns

True if current Mat has been allocated (by the constructor or therefore).

isMemoryOwner()

```
bool isMemoryOwner ( ) const  [inline]
```

brief Returns whether the Mat is the owner of the memory it access.
If not, the memory won't be freed if the Mat is destroyed.

Returns

True if the Mat is owning its memory, else false.

clone()

```
ERROR_CODE clone (
            const Mat & src )
```

Duplicates Mat by copy (deep copy).

Parameters

src	: the reference to the Mat to copy. This function copies the data array(s), it mark the new Mat as the memory owner.

move()

```
ERROR_CODE move (
            Mat & dst )
```

Moves Mat data to another Mat.

Parameters

dst	: the reference to the Mat to move. This function gives the attribute of the current Mat to the specified one. (No copy).

Note

: the current Mat is then no more usable since its loose its attributes.

swap()

```
static void swap (
            sl::Mat & mat1,
            sl::Mat & mat2 )  [static]
```

Swaps the content of the provided Mat (only swaps the pointers, no data copy).

Parameters

mat1	: the first mat.
mat2	: the second mat. This function swaps the pointers of the given Mat.

Member Data Documentation

name

```
sl::String name
```

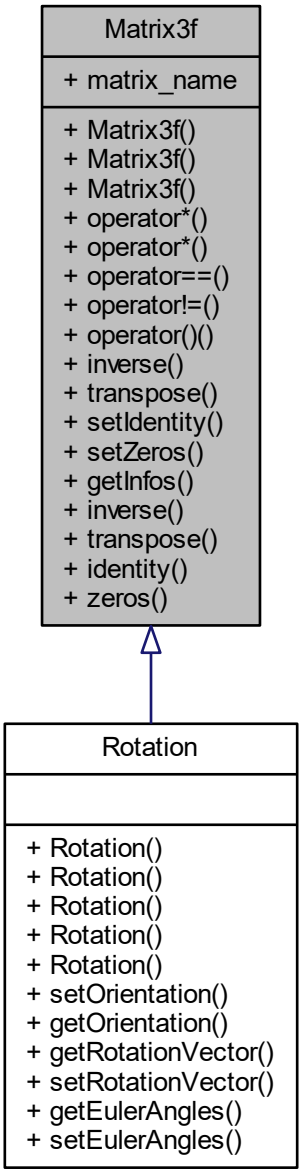
verbose

```
bool verbose = false
```

Matrix3f Class Reference

Represents a generic three-dimensional matrix.

Inheritance diagram for Matrix3f:



Public Member Functions

- Matrix3f ()

  Matrix3f default constructor.
- Matrix3f (float data[ ])

  Matrix3f copy constructor (deep copy).
- Matrix3f (const Matrix3f &mat)

  Matrix3f copy constructor (deep copy).
- Matrix3f operator∗ (const Matrix3f &mat) const

Gives the result of the multiplication between two Matrix3f.

- Matrix3f operator∗ (const double &scalar) const

  Gives the result of the multiplication between a Matrix3f and a scalar.
- bool operator== (const Matrix3f &mat) const

  Test two Matrix3f equality.
- bool operator!= (const Matrix3f &mat) const

  Test two Matrix3f inequality.
- float & operator() (int u, int v)

  Gets access to a specific point in the Matrix3f (read / write).
- void inverse ()

  Sets the Matrix3f to its inverse.
- void transpose ()

  Sets the RotationArray to its transpose.
- void setIdentity ()

  Sets the Matrix3f to identity.
- void setZeros ()

  Sets the Matrix3f to zero.
- sl::String getInfos ()

  Return the components of the Matrix3f in a sl::String.

Static Public Member Functions

- static Matrix3f inverse (const Matrix3f &rotation)

  Returns the inverse of a Matrix3f.
- static Matrix3f transpose (const Matrix3f &rotation)

  Returns the transpose of a Matrix3f.
- static Matrix3f identity ()

  Creates an identity Matrix3f.
- static Matrix3f zeros ()

  Creates a Matrix3f filled with zeros.

Public Attributes

- sl::String matrix_name

  Name of the matrix (optional).

Detailed Description

Represents a generic three-dimensional matrix.

It is defined in a row-major order, it means that, in the value buffer, the entire first row is stored first, followed by the entire second row, and so on. You can access the data with the 'r' ptr or by element attribute.

$$\begin{bmatrix} r00 & r01 & r02 \\ r10 & r11 & r22 \\ r20 & r21 & r22 \end{bmatrix}$$

Constructor & Destructor Documentation

Matrix3f() [1/3]

```
Matrix3f ( )
```

Matrix3f default constructor.

Matrix3f() [2/3]

```
Matrix3f (
            float data[] )
```

Matrix3f copy constructor (deep copy).

Matrix3f() [3/3]

```
Matrix3f (
            const Matrix3f & mat )
```

Matrix3f copy constructor (deep copy).

Parameters

rotation	: the Matrix3f to copy.

Member Function Documentation

operator∗() [1/2]

```
Matrix3f operator* (
            const Matrix3f & mat ) const
```

Gives the result of the multiplication between two Matrix3f.

operator∗() [2/2]

```
Matrix3f operator* (
            const double & scalar ) const
```

Gives the result of the multiplication between a Matrix3f and a scalar.

operator==()

```
bool operator== (
            const Matrix3f & mat ) const
```

Test two Matrix3f equality.

operator"!=()

```
bool operator!= (
            const Matrix3f & mat ) const
```

Test two Matrix3f inequality.

operator()()

```
float& operator() (
            int u,
```

```
            int v )
```

Gets access to a specific point in the Matrix3f (read / write).

Parameters

u	: specify the row
v	: specify the column

Returns

The value at the u, v coordinates.

inverse() [1/2]

```
void inverse ( )
```

Sets the Matrix3f to its inverse.

inverse() [2/2]

```
static Matrix3f inverse (
            const Matrix3f & rotation )  [static]
```

Returns the inverse of a Matrix3f.

Parameters

rotation	: the Matrix3f to compute the inverse from.

Returns

The inverse of the given Matrix3f

transpose() [1/2]

```
void transpose ( )
```

Sets the RotationArray to its transpose.

transpose() [2/2]

```
static Matrix3f transpose (
            const Matrix3f & rotation )  [static]
```

Returns the transpose of a Matrix3f.

Parameters

rotation	: the Matrix3f to compute the transpose from.

Returns

The transpose of the given Matrix3f

setIdentity()

```
void setIdentity ( )
```

Sets the Matrix3f to identity.

identity()

```
static Matrix3f identity ( )  [static]
```

Creates an identity Matrix3f.

Returns

A Matrix3f set to identity.

setZeros()

```
void setZeros ( )
```

Sets the Matrix3f to zero.

zeros()

```
static Matrix3f zeros ( )  [static]
```

Creates a Matrix3f filled with zeros.

Returns

A Matrix3f set to zero.

getInfos()

```
sl::String getInfos ( )
```

Return the components of the Matrix3f in a sl::String.

Returns

A sl::String containing the components of the current Matix3f.

Member Data Documentation

matrix_name

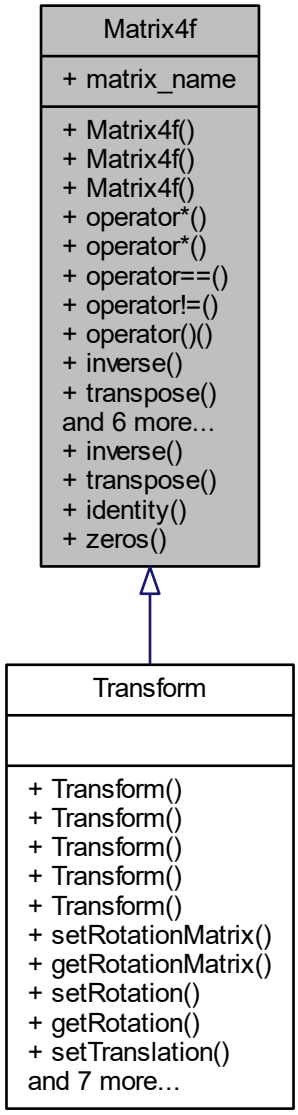
```
sl::String matrix_name
```

Name of the matrix (optional).

Matrix4f Class Reference

Represents a generic fourth-dimensional matrix.

Inheritance diagram for Matrix4f:



Public Member Functions

- Matrix4f ()

  Matrix4f default constructor.
- Matrix4f (float data[ ])

  Matrix4f copy constructor (deep copy).
- Matrix4f (const Matrix4f &mat)

  Matrix4f copy constructor (deep copy).
- Matrix4f operator∗ (const Matrix4f &mat) const
- Matrix4f operator∗ (const double &scalar) const
- bool operator== (const Matrix4f &mat) const

- bool operator!= (const Matrix4f &mat) const
- float & operator() (int u, int v)

  Gets access to a specific point in the Matrix4f (read / write).
- ERROR_CODE inverse ()

  Sets the Matrix4f to its inverse.
- void transpose ()

  Sets the Matrix4f to its transpose.
- void setIdentity ()

  Sets the Matrix4f to identity.
- void setZeros ()

  Sets the Matrix4f to zero.
- ERROR_CODE setSubMatrix3f (sl::Matrix3f input, int row=0, int column=0)

  Sets a 3x3 Matrix inside the Matrix4f.
- ERROR_CODE setSubVector3f (sl::Vector3< float > input, int column=3)

  Sets a 3x1 Vector inside the Matrix4f at the specified column index.
- ERROR_CODE setSubVector4f (sl::Vector4< float > input, int column=3)

  Sets a 4x1 Vector inside the Matrix4f at the specified column index.
- sl::String getInfos ()

  Return the components of the Matrix4f in a sl::String.

Static Public Member Functions

- static Matrix4f inverse (const Matrix4f &mat)

  Creates the inverse of a Matrix4f.
- static Matrix4f transpose (const Matrix4f &mat)

  Creates the transpose of a Matrix4f.
- static Matrix4f identity ()

  Creates an identity Matrix4f.
- static Matrix4f zeros ()

  Creates a Matrix4f filled with zeros.

Public Attributes

- sl::String matrix_name

  Name of the matrix (optional).

Detailed Description

Represents a generic fourth-dimensional matrix.

sIt is defined in a row-major order, it means that, in the value buffer, the entire first row is stored first, followed by the entire second row, and so on. You can access the data by the 'm' ptr or by the element attribute.

$$\begin{bmatrix} r00 & r01 & r02 & tx \\ r10 & r11 & r22 & ty \\ r20 & r21 & r22 & tz \\ m30 & m31 & m32 & m33 \end{bmatrix}$$

Constructor & Destructor Documentation

Matrix4f() [1/3]

```
Matrix4f ( )
```

Matrix4f default constructor.

Matrix4f() [2/3]

```
Matrix4f (
            float data[] )
```

Matrix4f copy constructor (deep copy).

Matrix4f() [3/3]

```
Matrix4f (
            const Matrix4f & mat )
```

Matrix4f copy constructor (deep copy).

Parameters

rotation	: the Matrix4f to copy.

Member Function Documentation

operator∗() [1/2]

```
Matrix4f operator* (
            const Matrix4f & mat ) const
```

brief Gives the result of the multiplication between two Matrix4f.

operator∗() [2/2]

```
Matrix4f operator* (
            const double & scalar ) const
```

brief Gives the result of the multiplication between a Matrix4f and a scalar.

operator==()

```
bool operator== (
            const Matrix4f & mat ) const
```

brief Test two Matrix4f equality.

operator"!=()

```
bool operator!= (
            const Matrix4f & mat ) const
```

brief Test two Matrix4f inequality.

operator()()

```
float& operator() (
            int u,
```

```
            int v )
```

Gets access to a specific point in the Matrix4f (read / write).

Parameters

u	: specify the row.
v	: specify the column.

Returns

The value at the u, v coordinates.

inverse() [1/2]

```
ERROR_CODE inverse ( )
```

Sets the Matrix4f to its inverse.

Returns

SUCCESS if the inverse has been computed, ERROR_CODE_FAILURE is not (det = 0).

inverse() [2/2]

```
static Matrix4f inverse (
            const Matrix4f & mat )  [static]
```

Creates the inverse of a Matrix4f.

Parameters

rotation	: the Matrix4f to compute the inverse from.

Returns

The inverse of the given Matrix4f.

transpose() [1/2]

```
void transpose ( )
```

Sets the Matrix4f to its transpose.

transpose() [2/2]

```
static Matrix4f transpose (
            const Matrix4f & mat )  [static]
```

Creates the transpose of a Matrix4f.

Parameters

rotation	: the Matrix4f to compute the transpose from.

Returns

The transpose of the given Matrix4f.

setIdentity()

```
void setIdentity ( )
```

Sets the Matrix4f to identity.

identity()

```
static Matrix4f identity ( )  [static]
```

Creates an identity Matrix4f.

Returns

A Matrix4f set to identity.

setZeros()

```
void setZeros ( )
```

Sets the Matrix4f to zero.

zeros()

```
static Matrix4f zeros ( )  [static]
```

Creates a Matrix4f filled with zeros.

Returns

A Matrix4f set to zero.

setSubMatrix3f()

```
ERROR_CODE setSubMatrix3f (
             sl::Matrix3f input,
             int row = 0,
             int column = 0 )
```

Sets a 3x3 Matrix inside the Matrix4f.

Note

Can be used to set the rotation matrix when the matrix4f is a pose or an isometric matrix.

Parameters

sl::Matrix3f	: sub matrix to put inside the Matrix4f.
row	: index of the row to start the 3x3 block. Must be 0 or 1.
column	: index of the column to start the 3x3 block. Must be 0 or 1.

Returns

SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

setSubVector3f()

```
ERROR_CODE setSubVector3f (
            sl::Vector3< float > input,
            int column = 3 )
```

Sets a 3x1 Vector inside the Matrix4f at the specified column index.

Note

Can be used to set the Translation/Position matrix when the matrix4f is a pose or an isometry.

Parameters

sl::Vector3	: sub vector to put inside the Matrix4f.
column	: index of the column to start the 3x3 block. By default, it is the last column (translation for a Pose).

Returns

SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

setSubVector4f()

```
ERROR_CODE setSubVector4f (
            sl::Vector4< float > input,
            int column = 3 )
```

Sets a 4x1 Vector inside the Matrix4f at the specified column index.

Note

Can be used to set the Translation/Position matrix when the matrix4f is a pose or an isometry.

Parameters

sl::Vector4	: sub vector to put inside the Matrix4f.
column	: index of the column to start the 3x3 block. By default, it is the last column (translation for a Pose).

Returns

SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

getInfos()

```
sl::String getInfos ( )
```

Return the components of the Matrix4f in a sl::String.

Returns

A sl::String containing the components of the current Matrix4f.

Member Data Documentation

matrix_name

```
sl::String matrix_name
```

Name of the matrix (optional).

Mesh Class Reference

A mesh contains the geometric (and optionally texturing) data of the scene computed by the spatial mapping.

Public Types

- typedef std::vector< size_t > chunkList

Public Member Functions

- Mesh ()

  Default constructor which creates an empty Mesh.
- ~Mesh ()

  Mesh destructor.
- sl::Chunk & operator[ ] (int index)
- size_t getNumberOfTriangles ()

  Compute the total number of triangles stored in all chunks.
- void updateMeshFromChunkList (chunkList IDs=chunkList(0))

  Update sl::Mesh::vertices / sl::Mesh::normals / sl::Mesh::triangles / sl::Mesh::uv from chunks' data pointed by the given chunkList.
- chunkList getVisibleList (sl::Transform camera_pose)

  Compute the list of visible chunk from a specific point of view.
- chunkList getSurroundingList (sl::Transform camera_pose, float radius)

  Compute the list of chunk which are close to a specific point of view.
- bool filter (MeshFilterParameters mesh_filter_params=MeshFilterParameters(), bool update_↩ mesh=true)

  Filters the mesh according to the given parameters.
- bool applyTexture (MESH_TEXTURE_FORMAT texture_format=MESH_TEXTURE_RGB)

  Applies texture to the mesh.
- bool save (sl::String filename, MESH_FILE_FORMAT type=MESH_FILE_OBJ, chunkList IDs=chunk↩ List(0))

  Saves the current Mesh into a file.
- bool load (sl::String filename, bool update_mesh=true)

  Loads the mesh from a file.
- void clear ()

  Clear all the data.

Public Attributes

- std::vector< sl::Chunk > chunks
- std::vector< sl::float3 > vertices
- std::vector< sl::uint3 > triangles
- std::vector< sl::float3 > normals
- std::vector< sl::float2 > uv
- Texture texture

Detailed Description

A mesh contains the geometric (and optionally texturing) data of the scene computed by the spatial mapping.

By default the mesh is defined by a set of Chunk, this way we update only the data that have to be updated avoiding a time consuming remapping process every time a small part of the Mesh is updated.

Member Typedef Documentation

chunkList

```
typedef std::vector<size_t> chunkList
```

Constructor & Destructor Documentation

Mesh()

```
Mesh ( )
```

Default constructor which creates an empty Mesh.

~Mesh()

```
~Mesh ( )
```

Mesh destructor.

Member Function Documentation

operator[]()

```
sl::Chunk& operator[] (
            int index )
```

define the [] operator to directly access the desired chunk.

getNumberOfTriangles()

```
size_t getNumberOfTriangles ( )
```

Compute the total number of triangles stored in all chunks.

Returns

The number of triangles stored in all chunks.

updateMeshFromChunkList()

```
void updateMeshFromChunkList (
            chunkList IDs = chunkList(0) )
```

Update sl::Mesh::vertices / sl::Mesh::normals / sl::Mesh::triangles / sl::Mesh::uv from chunks' data pointed by the given chunkList.

Parameters

IDs	: the index of chunks which will be concatenated. default : (empty).

Note

If the given chunkList is empty, all chunks will be used to update the current Mesh.

getVisibleList()

```
chunkList getVisibleList (
            sl::Transform camera_pose )
```

Compute the list of visible chunk from a specific point of view.

Parameters

world_reference_pose	: the point of view, given in world reference.

Returns

The list of visible chunks.

getSurroundingList()

```
chunkList getSurroundingList (
            sl::Transform camera_pose,
            float radius )
```

Compute the list of chunk which are close to a specific point of view.

Parameters

world_reference_position	: the point of view, given in world reference.
radius	: the radius in defined sl::UNIT.

Returns

The list of chunks close to the given point.

filter()

```
bool filter (
            MeshFilterParameters mesh_filter_params = MeshFilterParameters(),
            bool update_mesh = true )
```

Filters the mesh according to the given parameters.

Parameters

mesh_filter_params	: defines the filtering parameters, for more info checkout the sl::MeshFilterParameters documentation. default : preset.
update_mesh	: if set to true the mesh data (vertices/normals/triangles) are updated otherwise only the chunks data are updated. default : true.

Returns

True if the filtering was successful, false otherwise.

Note

The filtering is a costly operation but the resulting mesh is significantly lighter and less noisy.
Updating the Mesh is time consuming, consider using only Chunks for better performances.

applyTexture()

```
bool applyTexture (
            MESH_TEXTURE_FORMAT texture_format = MESH_TEXTURE_RGB )
```

Applies texture to the mesh.

Parameters

texture_format	: define the number of channel desired for the computed texture. default : MESH_TEXTURE_RGB.

Returns

True if the texturing was successful, false otherwise.

Warning

SpatialMappingParams::saveTextureData must be set as true when enabling the spatial mapping to be able to apply the textures.
The mesh should be filtered before calling this function since Mesh::filter will erased the textures, the texturing is also significantly slower on non-filtered meshes.

save()

```
bool save (
            sl::String filename,
            MESH_FILE_FORMAT type = MESH_FILE_OBJ,
            chunkList IDs = chunkList(0) )
```

Saves the current Mesh into a file.

Parameters

filename	: the path and filename of the mesh.
type	: defines the file type (extension). default : MESH_FILE_OBJ.
IDs	: (by default empty) Specify a set of chunks to be saved, if none provided alls chunks are saved. default : (empty).

Returns

True if the file was successfully saved, false otherwise.

Note

Only sl::MESH_FILE_OBJ support textures data.
This function operate on the Mesh not on the chunks. This way you can save different parts of your Mesh (update your Mesh with Mesh::updateMeshFromChunkList).

load()

```
bool load (
            sl::String filename,
            bool update_mesh = true )
```

Loads the mesh from a file.

Parameters

filename	: the path and filename of the mesh (do not forget the extension).
update_mesh	: if set to true the mesh data (vertices/normals/triangles) are updated otherwise only the chunks data are updated. default : true.

Returns

True if the loading was successful, false otherwise.

Note

Updating the Mesh is time consuming, consider using only Chunks for better performances.

clear()

```
void clear ( )
```

Clear all the data.

Member Data Documentation

chunks

```
std::vector<sl::Chunk> chunks
```

contains the list of chunks

vertices

```
std::vector<sl::float3> vertices
```

Vertices are defined by a 3D point {x,y,z}.

triangles

```
std::vector<sl::uint3> triangles
```

Triangles (or faces) contains the index of its three vertices. It corresponds to the 3 vertices of the triangle {v1, v2, v3}.

normals

`std::vector<sl::float3> normals`

Normals are defined by three components, {nx, ny, nz}. Normals are defined for each vertices.

uv

`std::vector<sl::float2> uv`

Texture coordinates defines 2D points on a texture.

Note

Contains data only if your mesh have textures (by loading it or calling sl::Mesh::applyTexture).

texture

`Texture texture`

texture.

Note

Contains data only if your mesh have textures (by loading it or calling sl::Mesh::applyTexture).

MeshFilterParameters Class Reference

Parameters for the optional filtering step of a sl::Mesh.
A default constructor is enabled and set to its default parameters.

Public Types

- enum FILTER { FILTER_LOW, FILTER_MEDIUM, FILTER_HIGH }

  List available mesh filtering intensity.

Public Member Functions

- MeshFilterParameters (FILTER filtering_=FILTER_LOW)

  Default constructor, set all parameters to their default and optimized values.
- void set (FILTER filtering_=FILTER_LOW)

  Sets the filtering intensity.
- bool save (sl::String filename)

  Saves the current bunch of parameters into a file.
- bool load (sl::String filename)

  Loads the values of the parameters contained in a file.

Public Attributes

- FILTER filtering = FILTER::FILTER_LOW

Detailed Description

Parameters for the optional filtering step of a sl::Mesh.
A default constructor is enabled and set to its default parameters.

Note

Parameters can be user adjusted.

Constructor & Destructor Documentation

MeshFilterParameters()

```
MeshFilterParameters (
            FILTER filtering_ = FILTER_LOW )  [inline]
```

Default constructor, set all parameters to their default and optimized values.

Member Function Documentation

set()

```
void set (
            FILTER filtering_ = FILTER_LOW )  [inline]
```

Sets the filtering intensity.

Parameters

*filtering_*	: the desired sl::MeshFilterParameters::FILTER.

save()

```
bool save (
            sl::String filename )
```

Saves the current bunch of parameters into a file.

Parameters

filename	: the path to the file in which the parameters will be stored.

Returns

true if the file was successfully saved, otherwise false.

load()

```
bool load (
            sl::String filename )
```

Loads the values of the parameters contained in a file.

Parameters

filename	: the path to the file from which the parameters will be loaded.

Returns

true if the file was successfully loaded, otherwise false.

Member Data Documentation

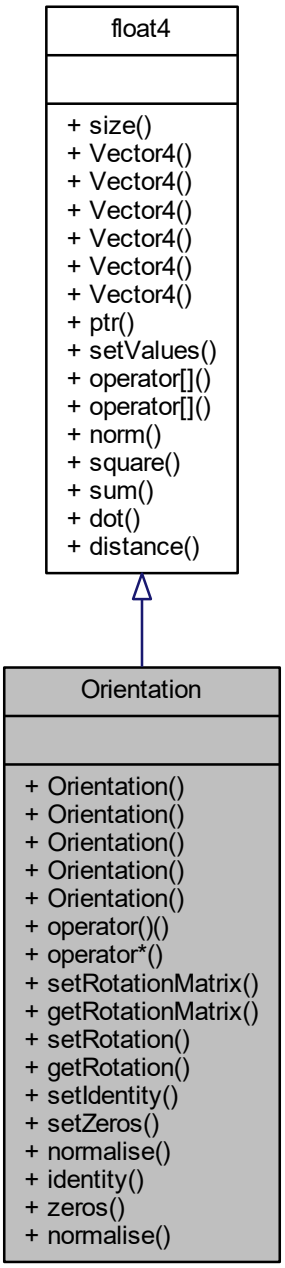
filtering

```
FILTER filtering = FILTER::FILTER_LOW
```

Orientation Class Reference

Designed to contain orientation (quaternion) data of the positional tracking.

Inheritance diagram for Orientation:



Public Member Functions

- Orientation ()

  empty Orientation default constructor.
- Orientation (const Orientation &orientation)

  Orientation copy constructor (deep copy).
- Orientation (const sl::float4 &in)

Orientation copy constructor (deep copy).

- Orientation (const Rotation &rotation)

  Orientation constructor from an Rotation.
- Orientation (const Translation &tr1, const Translation &tr2)

  Orientation constructor from a vector represented by two Translation.
- float operator() (int x)

  Returns the value at specific position in the Orientation.
- Orientation operator∗ (const Orientation &orientation) const

  Multiplication operator by an Orientation.
- void setRotationMatrix (const Rotation &rotation)

  Sets the orientation from a Rotation.
- Rotation getRotationMatrix () const

  Returns the current orientation as a Rotation.
- void setRotation (const Rotation &rotation)

  Alias to setRotationMatrix()
- Rotation getRotation () const

  Alias to getRotationMatrix()
- void setIdentity ()

  Sets the current Orientation to identity.
- void setZeros ()

  Fills the current Orientation with zeros.
- void normalise ()

  Normalizes the current Orientation.
- _FCT_CPU_GPU_ int size () const
- _FCT_CPU_GPU_ const T ∗ ptr () const
- _FCT_CPU_GPU_ Vector4< T > & setValues (const T ∗b)
- _FCT_CPU_GPU_ T & operator[ ] (int i)
- _FCT_CPU_GPU_ const T & operator[ ] (int i) const
- _FCT_CPU_GPU_ float norm ()

  returns the norm of the vector
- _FCT_CPU_GPU_ float square ()

  returns the squared norm of the vector
- _FCT_CPU_GPU_ float sum ()

  returns the sum of the vector

Static Public Member Functions

- static Orientation identity ()

  Creates an Orientation initialized to identity.
- static Orientation zeros ()

  Creates an Orientation filled with zeros.
- static Orientation normalise (const Orientation &orient)

  Creates the normalized version of an existing Orientation.
- static _FCT_CPU_GPU_ float dot (const Vector4< T > &a, const Vector4< T > &b)

  returns the dot product of two vector
- static _FCT_CPU_GPU_ float distance (const Vector4< T > &a, const Vector4< T > &b)

  returns the distance between two vector

Detailed Description

Designed to contain orientation (quaternion) data of the positional tracking.

sl::Orientation is a vector defined as [ox, oy, oz, ow].

Constructor & Destructor Documentation

Orientation() [1/5]

```
Orientation ( )
```

empty Orientation default constructor.

Orientation() [2/5]

```
Orientation (
            const Orientation & orientation )
```

Orientation copy constructor (deep copy).

Parameters

orientation	: the Orientation to copy.

Orientation() [3/5]

```
Orientation (
            const sl::float4 & in )
```

Orientation copy constructor (deep copy).

Parameters

in	: the vector to copy.

Orientation() [4/5]

```
Orientation (
            const Rotation & rotation )
```

Orientation constructor from an Rotation.

It converts the Rotation representation to the Orientation one.

Parameters

rotation	: the Rotation to be used.

Orientation() [5/5]

```
Orientation (
            const Translation & tr1,
            const Translation & tr2 )
```

Orientation constructor from a vector represented by two Translation.

Parameters

tr1	: the first point of the vector.
tr2	: the second point of the vector.

Member Function Documentation

operator()()

```
float operator() (
            int x )
```

Returns the value at specific position in the Orientation.

Parameters

x	: the position of the value

Returns

The value at the x position.

operator∗()

```
Orientation operator* (
            const Orientation & orientation ) const
```

Multiplication operator by an Orientation.

Parameters

orientation	: the orientation.

Returns

The current orientation after being multiplied by the other orientation.

setRotationMatrix()

```
void setRotationMatrix (
            const Rotation & rotation )
```

Sets the orientation from a Rotation.

Parameters

rotation	: the Rotation to be used.

getRotationMatrix()

```
Rotation getRotationMatrix ( ) const
```

Returns the current orientation as a Rotation.

Returns

The rotation computed from the orientation data.

setRotation()

```
void setRotation (
            const Rotation & rotation )  [inline]
```

Alias to setRotationMatrix()

Deprecated

getRotation()

```
Rotation getRotation ( ) const  [inline]
```

Alias to getRotationMatrix()

Deprecated

setIdentity()

```
void setIdentity ( )
```

Sets the current Orientation to identity.

identity()

```
static Orientation identity ( )  [static]
```

Creates an Orientation initialized to identity.

Returns

An identity Orientation.

setZeros()

```
void setZeros ( )
```

Fills the current Orientation with zeros.

zeros()

```
static Orientation zeros ( )  [static]
```

Creates an Orientation filled with zeros.

Returns

An Orientation filled with zeros.

normalise() [1/2]

```
void normalise ( )
```

Normalizes the current Orientation.

normalise() [2/2]

```
static Orientation normalise (
            const Orientation & orient )  [static]
```

Creates the normalized version of an existing Orientation.

Parameters

orient	: the Orientation to be used.

Returns

The normalized version of the Orientation.

size()

```
_FCT_CPU_GPU_ int size ( ) const  [inline], [inherited]
```

ptr()

```
_FCT_CPU_GPU_ const T* ptr ( ) const  [inline], [inherited]
```

setValues()

```
_FCT_CPU_GPU_ Vector4<T>& setValues (
            const T * b )  [inline], [inherited]
```

operator[]() [1/2]

```
_FCT_CPU_GPU_ T& operator[] (
            int i )  [inline], [inherited]
```

operator[]() [2/2]

```
_FCT_CPU_GPU_ const T& operator[] (
            int i ) const  [inline], [inherited]
```

norm()

```
_FCT_CPU_GPU_ float norm ( )  [inline], [inherited]
```

returns the norm of the vector

square()

```
_FCT_CPU_GPU_ float square ( )  [inline], [inherited]
```

returns the squared norm of the vector

sum()

```
_FCT_CPU_GPU_ float sum ( )  [inline], [inherited]
```

returns the sum of the vector

dot()

```
static _FCT_CPU_GPU_ float dot (
            const Vector4< T > & a,
            const Vector4< T > & b )  [inline], [static], [inherited]
```

returns the dot product of two vector

distance()

```
static _FCT_CPU_GPU_ float distance (
            const Vector4< T > & a,
            const Vector4< T > & b )  [inline], [static], [inherited]
```

returns the distance between two vector

Pose Class Reference

Contains the positional tracking data which gives the position and orientation of the ZED in 3D space.

Public Member Functions

- Pose ()

  Default constructor which creates an empty Pose (identity).
- Pose (const Pose &pose)

  Pose constructor with deep copy.
- Pose (const sl::Transform &pose_data, unsigned long long mtimestamp=0, int mconfidence=0)

  Pose constructor with deep copy.
- ~Pose ()

  Pose destructor.
- sl::Translation getTranslation ()

  Returns the camera translation from the pose.
- sl::Orientation getOrientation ()

  Returns the camera orientation from the pose.
- sl::Rotation getRotationMatrix ()

  Returns the camera rotation (3x3) from the pose.
- sl::Rotation getRotation ()

  Alias to getRotationMatrix()
- sl::float3 getRotationVector ()

  Returns the camera rotation (3x1 rotation vector obtained from 3x3 rotation matrix using Rodrigues formula) of the pose.
- sl::float3 getEulerAngles (bool radian=true)

  Convert the Rotation of the Transform as Euler angles.

Public Attributes

- sl::Transform pose_data
- unsigned long long timestamp
- int pose_confidence
- bool valid

Detailed Description

Contains the positional tracking data which gives the position and orientation of the ZED in 3D space.

Many representation of the position and orientation are enabled, like rotations, angles, as well as connected values, like timestamp, confidence.

Constructor & Destructor Documentation

Pose() [1/3]

```
Pose ( )
```

Default constructor which creates an empty Pose (identity).

Pose() [2/3]

```
Pose (
            const Pose & pose )
```

Pose constructor with deep copy.

Pose() [3/3]

```
Pose (
            const sl::Transform & pose_data,
            unsigned long long mtimestamp = 0,
            int mconfidence = 0 )
```

Pose constructor with deep copy.

~Pose()

```
~Pose ( )
```

Pose destructor.

Member Function Documentation

getTranslation()

```
sl::Translation getTranslation ( )
```

Returns the camera translation from the pose.

Returns

The translation vector of the ZED position.

getOrientation()

```
sl::Orientation getOrientation ( )
```

Returns the camera orientation from the pose.

Returns

The orientation vector of the ZED position.

getRotationMatrix()

```
sl::Rotation getRotationMatrix ( )
```

Returns the camera rotation (3x3) from the pose.

Returns

The rotation matrix of the ZED position.

getRotation()

```
sl::Rotation getRotation ( )  [inline]
```

Alias to getRotationMatrix()

Deprecated

getRotationVector()

```
sl::float3 getRotationVector ( )
```

Returns the camera rotation (3x1 rotation vector obtained from 3x3 rotation matrix using Rodrigues formula) of the pose.

Returns

The rotation vector of the ZED position.

getEulerAngles()

```
sl::float3 getEulerAngles (
            bool radian = true )
```

Convert the Rotation of the Transform as Euler angles.

Parameters

radian	: Define if the angle in is radian or degree. default : true.

Returns

The Euler angles, as a sl::float3 (x=roll, y=pitch, z=yaw)

Member Data Documentation

pose_data

```
sl::Transform pose_data
```

4x4 Matrix which contains the rotation (3x3) and the translation. Orientation is extracted from this transform as well.

timestamp

```
unsigned long long timestamp
```

Timestamp of the pose. This timestamp should be compared with the camera timestamp for synchronization.

pose_confidence

```
int pose_confidence
```

Confidence/Quality of the pose estimation for the target frame.
A confidence metric of the tracking [0-100], 0 means that the tracking is lost, 100 means that the tracking can be fully trusted.

valid

`bool valid`

boolean that indicates if tracking is activated or not. You should check that first if something wrong.

RecordingState Struct Reference

Recording structure that contains information about SVO.

Public Attributes

- bool status
- double current_compression_time
- double current_compression_ratio
- double average_compression_time
- double average_compression_ratio

Detailed Description

Recording structure that contains information about SVO.

Member Data Documentation

status

`bool status`

status of current frame. May be true for success or false if frame could not be written in the SVO file.

current_compression_time

`double current_compression_time`

compression time for the current frame in ms.

current_compression_ratio

`double current_compression_ratio`

compression ratio (% of raw size) for the current frame.

average_compression_time

`double average_compression_time`

average compression time in ms since beginning of recording.

average_compression_ratio

`double average_compression_ratio`

compression ratio (% of raw size) since beginning of recording.

Resolution Struct Reference

Width and height of an array.

Public Member Functions

- Resolution (size_t w_=0, size_t h_=0)
- size_t area ()

  Returns the area of the image.
- bool operator== (const Resolution &that) const

  Tests if the given sl::Resolution has the same properties.
- bool operator!= (const Resolution &that) const

  Tests if the given sl::Resolution has different properties.

Public Attributes

- size_t width
- size_t height

Detailed Description

Width and height of an array.

Constructor & Destructor Documentation

Resolution()

```
Resolution (
             size_t w_ = 0,
             size_t h_ = 0 )  [inline]
```

Member Function Documentation

area()

```
size_t area ( )  [inline]
```

Returns the area of the image.

Returns

The number of pixels of the array.

operator==()

```
bool operator== (
             const Resolution & that ) const  [inline]
```

Tests if the given sl::Resolution has the same properties.

Returns

True if the sizes matches.

operator"!=()

```
bool operator!= (
            const Resolution & that ) const  [inline]
```

Tests if the given sl::Resolution has different properties.

Returns

True if the sizes are not equal.

Member Data Documentation

width

```
size_t width
```

array width in pixels

Referenced by Mat::getWidth(), Mat::getWidthBytes(), Resolution::operator!=(), and Resolution↩ ::operator==().

height

```
size_t height
```

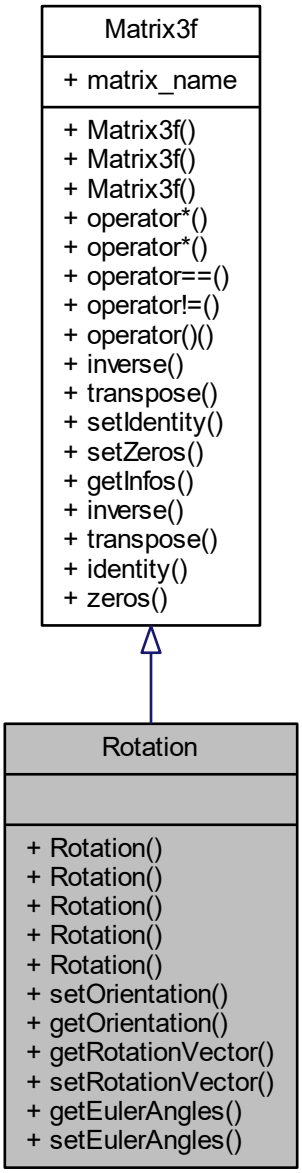
array height in pixels

Referenced by Resolution::area(), Mat::getHeight(), Resolution::operator!=(), and Resolution::operator==().

Rotation Class Reference

Designed to contain rotation data of the positional tracking. It inherits from the generic sl::Matrix3f.

Inheritance diagram for Rotation:



Public Member Functions

- Rotation ()

  empty Rotation default constructor.
- Rotation (const Rotation &rotation)

  Rotation copy constructor (deep copy).
- Rotation (const Matrix3f &mat)

  Rotation copy constructor (deep copy).
- Rotation (const Orientation &orientation)

Rotation constructor from an Orientation.

- Rotation (const float angle, const Translation &axis)

  Creates a Rotation representing the 3D rotation of angle around an arbitrary 3D axis.
- void setOrientation (const Orientation &orientation)

  Sets the Rotation from an Orientation.
- Orientation getOrientation () const

  Returns the Orientation corresponding to the current Rotation.
- sl::float3 getRotationVector ()

  Returns the 3x1 rotation vector obtained from 3x3 rotation matrix using Rodrigues formula.
- void setRotationVector (const sl::float3 &vec_rot)

  Sets the Rotation from a rotation vector (using Rodrigues' transformation).
- sl::float3 getEulerAngles (bool radian=true) const

  Convert the Rotation as Euler angles.
- void setEulerAngles (const sl::float3 &euler_angles, bool radian=true)

  Sets the Rotation from the Euler angles.
- Matrix3f operator∗ (const Matrix3f &mat) const

  Gives the result of the multiplication between two Matrix3f.
- Matrix3f operator∗ (const double &scalar) const

  Gives the result of the multiplication between a Matrix3f and a scalar.
- bool operator== (const Matrix3f &mat) const

  Test two Matrix3f equality.
- bool operator!= (const Matrix3f &mat) const

  Test two Matrix3f inequality.
- float & operator() (int u, int v)

  Gets access to a specific point in the Matrix3f (read / write).
- void inverse ()

  Sets the Matrix3f to its inverse.
- void transpose ()

  Sets the RotationArray to its transpose.
- void setIdentity ()

  Sets the Matrix3f to identity.
- void setZeros ()

  Sets the Matrix3f to zero.
- sl::String getInfos ()

  Return the components of the Matrix3f in a sl::String.

Static Public Member Functions

- static Matrix3f inverse (const Matrix3f &rotation)

  Returns the inverse of a Matrix3f.
- static Matrix3f transpose (const Matrix3f &rotation)

  Returns the transpose of a Matrix3f.
- static Matrix3f identity ()

  Creates an identity Matrix3f.
- static Matrix3f zeros ()

  Creates a Matrix3f filled with zeros.

Public Attributes

- sl::String matrix_name

  Name of the matrix (optional).

Detailed Description

Designed to contain rotation data of the positional tracking. It inherits from the generic sl::Matrix3f.

Constructor & Destructor Documentation

Rotation() [1/5]

```
Rotation ( )
```

empty Rotation default constructor.

Rotation() [2/5]

```
Rotation (
            const Rotation & rotation )
```

Rotation copy constructor (deep copy).

Parameters

rotation	: the Rotation to copy.

Rotation() [3/5]

```
Rotation (
            const Matrix3f & mat )
```

Rotation copy constructor (deep copy).

Parameters

mat	: the mat to copy.

Rotation() [4/5]

```
Rotation (
            const Orientation & orientation )
```

Rotation constructor from an Orientation.

It converts the Orientation representation to the Rotation one.

Parameters

orientation	: the Orientation to be used.

Rotation() [5/5]

```
Rotation (
            const float angle,
            const Translation & axis )
```

Creates a Rotation representing the 3D rotation of angle around an arbitrary 3D axis.

Parameters

angle	: the rotation angle in rad.
axis	: the 3D axis to rotate around.

Member Function Documentation

setOrientation()

```
void setOrientation (
            const Orientation & orientation )
```

Sets the Rotation from an Orientation.

Parameters

orientation	: the Orientation containing the rotation to set.

getOrientation()

```
Orientation getOrientation ( ) const
```

Returns the Orientation corresponding to the current Rotation.

Returns

The rotation of the current orientation.

getRotationVector()

```
sl::float3 getRotationVector ( )
```

Returns the 3x1 rotation vector obtained from 3x3 rotation matrix using Rodrigues formula.

Returns

The rotation vector.

setRotationVector()

```
void setRotationVector (
            const sl::float3 & vec_rot )
```

Sets the Rotation from a rotation vector (using Rodrigues' transformation).

Parameters

vec_rot	: the Rotation Vector.

getEulerAngles()

```
sl::float3 getEulerAngles (
            bool radian = true ) const
```

Convert the Rotation as Euler angles.

Parameters

radian	: Define if the angle in is radian or degree

Returns

The Euler angles, as a sl::float3 (x=roll, y=pitch, z=yaw)

setEulerAngles()

```
void setEulerAngles (
            const sl::float3 & euler_angles,
            bool radian = true )
```

Sets the Rotation from the Euler angles.

Parameters

euler_angles	: The Euler angles, as a sl::float3 (x=roll, y=pitch, z=yaw)
radian	: Define if the angle in is radian or degree

operator∗() [1/2]

```
Matrix3f operator* (
            const Matrix3f & mat ) const  [inherited]
```

Gives the result of the multiplication between two Matrix3f.

operator∗() [2/2]

```
Matrix3f operator* (
            const double & scalar ) const  [inherited]
```

Gives the result of the multiplication between a Matrix3f and a scalar.

operator==()

```
bool operator== (
            const Matrix3f & mat ) const  [inherited]
```

Test two Matrix3f equality.

operator"!=()

```
bool operator!= (
            const Matrix3f & mat ) const  [inherited]
```

Test two Matrix3f inequality.

operator()()

```
float& operator() (
            int u,
            int v )  [inherited]
```

Gets access to a specific point in the Matrix3f (read / write).

Parameters

Parameters

u	: specify the row
v	: specify the column

Returns

The value at the u, v coordinates.

inverse() [1/2]

```
void inverse ( )  [inherited]
```

Sets the Matrix3f to its inverse.

inverse() [2/2]

```
static Matrix3f inverse (
            const Matrix3f & rotation )  [static], [inherited]
```

Returns the inverse of a Matrix3f.

Parameters

rotation	: the Matrix3f to compute the inverse from.

Returns

The inverse of the given Matrix3f

transpose() [1/2]

```
void transpose ( )  [inherited]
```

Sets the RotationArray to its transpose.

transpose() [2/2]

```
static Matrix3f transpose (
            const Matrix3f & rotation )  [static], [inherited]
```

Returns the transpose of a Matrix3f.

Parameters

rotation	: the Matrix3f to compute the transpose from.

Returns

The transpose of the given Matrix3f

setIdentity()

```
void setIdentity ( )  [inherited]
```

Sets the Matrix3f to identity.

identity()

```
static Matrix3f identity ( )  [static], [inherited]
```

Creates an identity Matrix3f.

Returns

A Matrix3f set to identity.

setZeros()

```
void setZeros ( )  [inherited]
```

Sets the Matrix3f to zero.

zeros()

```
static Matrix3f zeros ( )  [static], [inherited]
```

Creates a Matrix3f filled with zeros.

Returns

A Matrix3f set to zero.

getInfos()

```
sl::String getInfos ( )  [inherited]
```

Return the components of the Matrix3f in a sl::String.

Returns

A sl::String containing the components of the current Matix3f.

Member Data Documentation

matrix_name

```
sl::String matrix_name  [inherited]
```

Name of the matrix (optional).

RuntimeParameters Class Reference

Parameters that defines the behavior of the sl::Camera::grab().

Public Member Functions

- RuntimeParameters (SENSING_MODE sensing_mode_=SENSING_MODE::SENSING_MODE_↩ STANDARD, bool enable_depth_=true, bool enable_point_cloud_=true, REFERENCE_FRAME measure3D_reference_frame_=REFERENCE_FRAME_CAMERA)

  Default constructor, set all parameters to their default and optimized values.

- bool save (sl::String filename)

  Saves the current bunch of parameters into a file.
- bool load (sl::String filename)

  Loads the values of the parameters contained in a file.

Public Attributes

- SENSING_MODE sensing_mode
- REFERENCE_FRAME measure3D_reference_frame
- bool enable_depth
- bool enable_point_cloud

Detailed Description

Parameters that defines the behavior of the sl::Camera::grab().

Default values are enabled.
You can customize it to fit your application and then save it to create a preset that can be loaded for further executions.

Constructor & Destructor Documentation

RuntimeParameters()

```
RuntimeParameters (
            SENSING_MODE sensing_mode_ = SENSING_MODE::SENSING_MODE_STANDARD,
            bool enable_depth_ = true,
            bool enable_point_cloud_ = true,
            REFERENCE_FRAME measure3D_reference_frame_ = REFERENCE_FRAME_CAMERA )  [inline]
```

Default constructor, set all parameters to their default and optimized values.

Member Function Documentation

save()

```
bool save (
            sl::String filename )
```

Saves the current bunch of parameters into a file.

Parameters

filename	: the path to the file in which the parameters will be stored.

Returns

true if the file was successfully saved, otherwise false.

load()

```
bool load (
            sl::String filename )
```

Loads the values of the parameters contained in a file.

Parameters

filename	: the path to the file from which the parameters will be loaded.

Returns

true if the file was successfully loaded, otherwise false.

Member Data Documentation

sensing_mode

```
SENSING_MODE sensing_mode
```

Defines the algorithm used for depth map computation, more info : sl::SENSING_MODE definition.
default : sl::SENSING_MODE::SENSING_MODE_STANDARD

measure3D_reference_frame

```
REFERENCE_FRAME measure3D_reference_frame
```

Provides 3D measures (point cloud and normals) in the desired reference frame (default is REFERENCE↩
_FRAME_CAMERA)
default : sl:REFERENCE_FRAME::REFERENCE_FRAME_CAMERA

Note

: replaces previous move_point_cloud_to_world_frame parameter.

enable_depth

```
bool enable_depth
```

Defines if the depth map should be computed.
If false, only the images are available.
default : true

enable_point_cloud

```
bool enable_point_cloud
```

Defines if the point cloud should be computed (including XYZRGBA).
default : true

Deprecated : this parameter is deprecated as of ZED SDK 2.1. Point cloud is now enabled once depth is.

SpatialMappingParameters Class Reference

Sets the spatial mapping parameters.

Public Types

- enum RESOLUTION { RESOLUTION_HIGH, RESOLUTION_MEDIUM, RESOLUTION_LOW }

List the spatial mapping resolution presets.

- enum RANGE { RANGE_NEAR, RANGE_MEDIUM, RANGE_FAR }

  List the spatial mapping depth range presets.
- typedef std::pair< float, float > interval

Public Member Functions

- SpatialMappingParameters (RESOLUTION resolution=RESOLUTION_HIGH, RANGE range=RANGE_MEDIUM, int max_memory_usage_=2048, bool save_texture_=true, bool keep_mesh_consistent_=true, bool inverse_triangle_vertices_order_=false)

  Default constructor, set all parameters to their default and optimized values.
- void set (RESOLUTION resolution=RESOLUTION_HIGH)

  Sets the resolution corresponding to the given sl::SpatialMappingParameters::RESOLUTION preset.
- void set (RANGE range=RANGE_MEDIUM)

  Sets the maximum value of depth corresponding to the given sl::SpatialMappingParameters::RANGE presets.
- bool save (sl::String filename)

  Saves the current bunch of parameters into a file.
- bool load (sl::String filename)

  Loads the values of the parameters contained in a file.

Static Public Member Functions

- static float get (RESOLUTION resolution=RESOLUTION_HIGH)

  Return the resolution corresponding to the given sl::SpatialMappingParameters::RESOLUTION preset.
- static float get (RANGE range=RANGE_MEDIUM)

  Return the maximum value of depth corresponding to the given sl::SpatialMappingParameters::RANGE presets.

Public Attributes

- float resolution_meter = 0.03f

  Spatial mapping resolution in meters, should fit interval::allowed_resolution.
- const interval allowed_resolution = std::make_pair(0.01f, 0.2f)

  The resolutions allowed by the spatial mapping.
- interval range_meter = std::make_pair(0.7f, 5.f)

  Depth integration range in meters.
- const interval allowed_min = std::make_pair(0.3f, 10.f)

  The minimal depth value allowed by the spatial mapping.
- const interval allowed_max = std::make_pair(2.f, 20.f)

  The maximal depth value allowed by the spatial mapping.
- bool save_texture = true

  Set to true if you want be able to apply texture to your mesh after its creation.
- bool keep_mesh_consistent = true

  Set to true if you want keep consistency between the mesh and its inner chunks data.
- int max_memory_usage = 2048

  The maximum CPU memory (in mega bytes) allocated for the meshing process (will fit your configuration in any case).
- bool inverse_triangle_vertices_order = false

  Specify if the order of the vertices of the triangles needs to be inverted. If your display process do not handle front and back face culling you can use this to set it right.

Detailed Description

Sets the spatial mapping parameters.

A default constructor is enabled and set to its default parameters.
You can customize it to fit your application and then save it to create a preset that can be loaded for further executions.

Note

Parameters can be user adjusted.

Member Typedef Documentation

interval

```
typedef std::pair<float, float> interval
```

Constructor & Destructor Documentation

SpatialMappingParameters()

```
SpatialMappingParameters (
            RESOLUTION resolution = RESOLUTION_HIGH,
            RANGE range = RANGE_MEDIUM,
            int max_memory_usage_ = 2048,
            bool save_texture_ = true,
            bool keep_mesh_consistent_ = true,
            bool inverse_triangle_vertices_order_ = false )  [inline]
```

Default constructor, set all parameters to their default and optimized values.

Member Function Documentation

get() [1/2]

```
static float get (
            RESOLUTION resolution = RESOLUTION_HIGH )  [inline], [static]
```

Return the resolution corresponding to the given sl::SpatialMappingParameters::RESOLUTION preset.

Parameters

resolution	: the desired sl::SpatialMappingParameters::RESOLUTION. default : RESOLUTION_HIGH.

Returns

The resolution in meter.

set() [1/2]

```
void set (
            RESOLUTION resolution = RESOLUTION_HIGH )  [inline]
```

Sets the resolution corresponding to the given sl::SpatialMappingParameters::RESOLUTION preset.

Parameters

resolution	: the desired sl::SpatialMappingParameters::RESOLUTION. default : RESOLUTION_HIGH.

get() [2/2]

```
static float get (
            RANGE range = RANGE_MEDIUM )  [inline], [static]
```

Return the maximum value of depth corresponding to the given sl::SpatialMappingParameters::RANGE presets.

Parameters

range	: the desired sl::SpatialMappingParameters::RANGE. default : RANGE_MEDIUM.

Returns

The maximum value of depth.

set() [2/2]

```
void set (
            RANGE range = RANGE_MEDIUM )  [inline]
```

Sets the maximum value of depth corresponding to the given sl::SpatialMappingParameters::RANGE presets.

Parameters

range	: the desired sl::SpatialMappingParameters::RANGE. default : RANGE_MEDIUM.

save()

```
bool save (
            sl::String filename )
```

Saves the current bunch of parameters into a file.

Parameters

filename	: the path to the file in which the parameters will be stored.

Returns

true if the file was successfully saved, otherwise false.

load()

```
bool load (
            sl::String filename )
```

Loads the values of the parameters contained in a file.

Parameters

filename	: the path to the file from which the parameters will be loaded.

Returns

true if the file was successfully loaded, otherwise false.

Member Data Documentation

resolution_meter

```
float resolution_meter = 0.03f
```

Spatial mapping resolution in meters, should fit interval::allowed_resolution.

allowed_resolution

```
const interval allowed_resolution = std::make_pair(0.01f, 0.2f)
```

The resolutions allowed by the spatial mapping.

range_meter

```
interval range_meter = std::make_pair(0.7f, 5.f)
```

Depth integration range in meters.

range_meter.first should fit interval::allowed_min.
range_meter.first will be set to sl::Camera::getDepthMinRangeValue if you do not change it.
range_meter.second should fit interval::allowed_max.

allowed_min

```
const interval allowed_min = std::make_pair(0.3f, 10.f)
```

The minimal depth value allowed by the spatial mapping.

allowed_max

```
const interval allowed_max = std::make_pair(2.f, 20.f)
```

The maximal depth value allowed by the spatial mapping.

save_texture

```
bool save_texture = true
```

Set to true if you want be able to apply texture to your mesh after its creation.

Note

This option will take more memory.

keep_mesh_consistent

```
bool keep_mesh_consistent = true
```

Set to true if you want keep consistency between the mesh and its inner chunks data.

Note

Updating the Mesh is time consuming, consider using only Chunks data for better performances.

max_memory_usage

```
int max_memory_usage = 2048
```

The maximum CPU memory (in mega bytes) allocated for the meshing process (will fit your configuration in any case).

inverse_triangle_vertices_order

```
bool inverse_triangle_vertices_order = false
```

Specify if the order of the vertices of the triangles needs to be inverted. If your display process do not handle front and back face culling you can use this to set it right.

String Class Reference

Defines a string.

Public Member Functions

- String ()
- ~String ()
- String (const String &str)
- String (const char *data)
- void set (const char *data)
- const char * get () const
- bool empty () const
- String & operator= (const String &str1)
- String & operator= (const char *data)
- operator const char * ()
- const char * c_str () const
- size_t size ()
- void clear ()

Detailed Description

Defines a string.

Constructor & Destructor Documentation

String() [1/3]

```
String ( )
```

~String()

```
~String ( )
```

String() [2/3]

```
String (
            const String & str )
```

String() [3/3]

```
String (
            const char * data )
```

Member Function Documentation

set()

```
void set (
            const char * data )
```

get()

```
const char* get ( ) const
```

empty()

```
bool empty ( ) const
```

operator=() [1/2]

```
String& operator= (
            const String & str1 )
```

operator=() [2/2]

```
String& operator= (
            const char * data )
```

operator const char *()

```
operator const char * ( )  [inline]
```

c_str()

```
const char* c_str ( ) const
```

size()

```
size_t size ( )
```

clear()

```
void clear ( )
```

Texture Class Reference

Contains information about texture image associated to a sl::Mesh.

Public Member Functions

- Texture ()
  Default constructor which creates an empty sl::Texture.
- ~Texture ()
  sl::Texture destructor.
- void clear ()
  Clears datas.

Public Attributes

- sl::Mat data
- unsigned int indice_gl
- sl::String name

Detailed Description

Contains information about texture image associated to a sl::Mesh.

Constructor & Destructor Documentation

Texture()

```
Texture ( )
```

Default constructor which creates an empty sl::Texture.

~Texture()

```
~Texture ( )
```

sl::Texture destructor.

Member Function Documentation

clear()

```
void clear ( )
```

Clears datas.

Member Data Documentation

data

```
sl::Mat data
```

sl::Mat that contains the data of the texture.

indice_gl

```
unsigned int indice_gl
```

useful for openGL binding reference (value not set by the SDK).

name

```
sl::String name
```

The name of the file in which the texture is saved.

TrackingParameters Class Reference

Parameters for ZED tracking initialization.

Public Member Functions

- TrackingParameters (sl::Transform init_pos=sl::Transform(), bool _enable_memory=true, sl::String ↩ _area_path=sl::String())

  Default constructor, set all parameters to their default and optimized values.
- bool save (sl::String filename)

  Saves the current bunch of parameters into a file.
- bool load (sl::String filename)

  Loads the values of the parameters contained in a file.

Public Attributes

- sl::Transform initial_world_transform
- bool enable_spatial_memory
- sl::String area_file_path

Detailed Description

Parameters for ZED tracking initialization.

A default constructor is enabled and set to its default parameters.
You can customize it to fit your application and then save it to create a preset that can be loaded for further executions.

Note

Parameters can be user adjusted.

Constructor & Destructor Documentation

TrackingParameters()

```
TrackingParameters (
            sl::Transform init_pos = sl::Transform(),
            bool _enable_memory = true,
            sl::String _area_path = sl::String() )  [inline]
```

Default constructor, set all parameters to their default and optimized values.

Member Function Documentation

save()

```
bool save (
            sl::String filename )
```

Saves the current bunch of parameters into a file.

Parameters

filename	: the path to the file in which the parameters will be stored.

Returns

true if the file was successfully saved, otherwise false.

load()

```
bool load (
            sl::String filename )
```

Loads the values of the parameters contained in a file.

Parameters

filename	: the path to the file from which the parameters will be loaded.

Returns

true if the file was successfully loaded, otherwise false.

Member Data Documentation

initial_world_transform

```
sl::Transform initial_world_transform
```

Position of the camera in the world frame when camera is started. By default it should be identity.
Use this sl::Transform to place the camera frame in the world frame.
default : Identity matrix

Note

The camera frame (defines the reference frame for the camera) is by default positioned at the world frame when tracking is started.

enable_spatial_memory

```
bool enable_spatial_memory
```

This mode enables the camera to learn and remember its surroundings. This helps correct motion tracking drift and position different cameras relative to each other in space.
default : true

Warning

: This mode requires few resources to run and greatly improves tracking accuracy. We recommend to leave it on by default.

area_file_path

```
sl::String area_file_path
```

Area localization mode can record and load (if areaFilePath is specified) a file that describes the surroundings.
default : (empty)

Note

Loading an area file will start a searching phase during which the camera will try to position itself in the previously learned area.

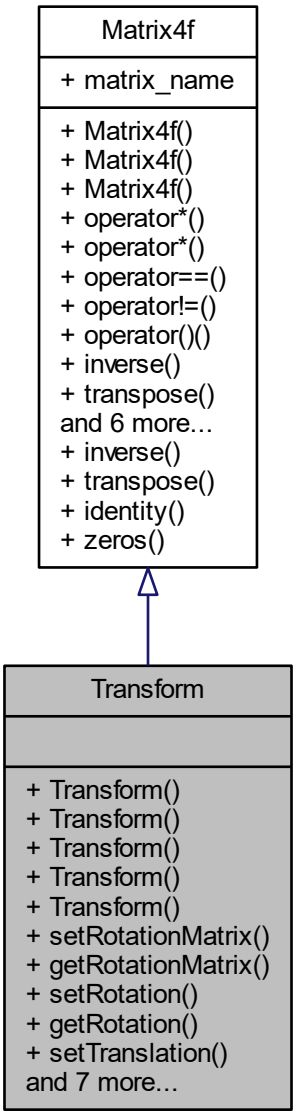
Warning

: The area file describes a specific location. If you are using an area file describing a different location, the tracking function will continuously search for a position and may not find a correct one.
The '.area' file can only be used with the same depth mode (sl::MODE) as the one used during area recording.

Transform Class Reference

Designed to contain translation and rotation data of the positional tracking.

Inheritance diagram for Transform:



Public Member Functions

- Transform ()
  Transform default constructor.
- Transform (const Transform &motion)
  Transform copy constructor (deep copy).
- Transform (const Matrix4f &mat)
  Transform copy constructor (deep copy).
- Transform (const Rotation &rotation, const Translation &translation)
  Transform constructor from a Rotation and a Translation.
- Transform (const Orientation &orientation, const Translation &translation)

Transform constructor from an Orientation and a Translation.

- void setRotationMatrix (const Rotation &rotation)

  Sets the rotation of the current Transform from an Rotation.
- Rotation getRotationMatrix () const

  Returns the Rotation of the current Transform.
- void setRotation (const Rotation &rotation)

  Alias to setRotationMatrix()
- Rotation getRotation () const

  Alias to getRotationMatrix()
- void setTranslation (const Translation &translation)

  Sets the translation of the current Transform from an Translation.
- Translation getTranslation () const

  Returns the Translation of the current Transform.
- void setOrientation (const Orientation &orientation)

  Sets the orientation of the current Transform from an Orientation.
- Orientation getOrientation () const

  Returns the Orientation of the current Transform.
- sl::float3 getRotationVector ()

  Returns the 3x1 rotation vector obtained from 3x3 rotation matrix using Rodrigues formula.
- void setRotationVector (const sl::float3 &vec_rot)

  Sets the Rotation 3x3 of the Transform with a 3x1 rotation vector (using Rodrigues' transformation).
- sl::float3 getEulerAngles (bool radian=true) const

  Convert the Rotation of the Transform as Euler angles.
- void setEulerAngles (const sl::float3 &euler_angles, bool radian=true)

  Sets the Rotation of the Transform from the Euler angles.
- Matrix4f operator∗ (const Matrix4f &mat) const
- Matrix4f operator∗ (const double &scalar) const
- bool operator== (const Matrix4f &mat) const
- bool operator!= (const Matrix4f &mat) const
- float & operator() (int u, int v)

  Gets access to a specific point in the Matrix4f (read / write).
- ERROR_CODE inverse ()

  Sets the Matrix4f to its inverse.
- void transpose ()

  Sets the Matrix4f to its transpose.
- void setIdentity ()

  Sets the Matrix4f to identity.
- void setZeros ()

  Sets the Matrix4f to zero.
- ERROR_CODE setSubMatrix3f (sl::Matrix3f input, int row=0, int column=0)

  Sets a 3x3 Matrix inside the Matrix4f.
- ERROR_CODE setSubVector3f (sl::Vector3< float > input, int column=3)

  Sets a 3x1 Vector inside the Matrix4f at the specified column index.
- ERROR_CODE setSubVector4f (sl::Vector4< float > input, int column=3)

  Sets a 4x1 Vector inside the Matrix4f at the specified column index.
- sl::String getInfos ()

  Return the components of the Matrix4f in a sl::String.

Static Public Member Functions

- static Matrix4f inverse (const Matrix4f &mat)

  Creates the inverse of a Matrix4f.
- static Matrix4f transpose (const Matrix4f &mat)

  Creates the transpose of a Matrix4f.
- static Matrix4f identity ()

  Creates an identity Matrix4f.
- static Matrix4f zeros ()

  Creates a Matrix4f filled with zeros.

Public Attributes

- sl::String matrix_name

  Name of the matrix (optional).

Detailed Description

Designed to contain translation and rotation data of the positional tracking.

It contains the orientation as well. It can be used to create any type of Matrix4x4 or sl::Matrix4f that must be specifically used for handling a rotation and position information (OpenGL, Tracking...) It inherits from the generic sl::Matrix4f

Constructor & Destructor Documentation

Transform() [1/5]

```
Transform ( )
```

Transform default constructor.

Transform() [2/5]

```
Transform (
            const Transform & motion )
```

Transform copy constructor (deep copy).

Parameters

motion	: the Transform to copy.

Transform() [3/5]

```
Transform (
            const Matrix4f & mat )
```

Transform copy constructor (deep copy).

Parameters

mat	: the Matrix4f to copy.

Transform() [4/5]

```
Transform (
            const Rotation & rotation,
            const Translation & translation )
```

Transform constructor from a Rotation and a Translation.

Parameters

rotation	: the Rotation to copy.
translation	: the Translation to copy.

Transform() [5/5]

```
Transform (
            const Orientation & orientation,
            const Translation & translation )
```

Transform constructor from an Orientation and a Translation.

Parameters

orientation	: the Orientation to copy.
translation	: the Translation to copy.

Member Function Documentation

setRotationMatrix()

```
void setRotationMatrix (
            const Rotation & rotation )
```

Sets the rotation of the current Transform from an Rotation.

Parameters

rotation	: the Rotation to be used.

getRotationMatrix()

```
Rotation getRotationMatrix ( ) const
```

Returns the Rotation of the current Transform.

Returns

The Rotation created from the Transform values.

Warning

The given Rotation contains a copy of the Transform values. Not references.

setRotation()

```
void setRotation (
            const Rotation & rotation )  [inline]
```

Alias to setRotationMatrix()

Deprecated

getRotation()

```
Rotation getRotation ( ) const  [inline]
```

Alias to getRotationMatrix()

Deprecated

setTranslation()

```
void setTranslation (
            const Translation & translation )
```

Sets the translation of the current Transform from an Translation.

Parameters

translation	: the Translation to be used.

getTranslation()

```
Translation getTranslation ( ) const
```

Returns the Translation of the current Transform.

Returns

The Translation created from the Transform values.

Warning

The given Translation contains a copy of the Transform values. Not references.

setOrientation()

```
void setOrientation (
            const Orientation & orientation )
```

Sets the orientation of the current Transform from an Orientation.

Parameters

orientation	: the Orientation to be used.

getOrientation()

```
Orientation getOrientation ( ) const
```

Returns the Orientation of the current Transform.

Returns

The Orientation created from the Transform values.

Warning

The given Orientation contains a copy of the Transform values. Not references.

getRotationVector()

```
sl::float3 getRotationVector ( )
```

Returns the 3x1 rotation vector obtained from 3x3 rotation matrix using Rodrigues formula.

Returns

The rotation vector.

setRotationVector()

```
void setRotationVector (
            const sl::float3 & vec_rot )
```

Sets the Rotation 3x3 of the Transform with a 3x1 rotation vector (using Rodrigues' transformation).

Parameters

vec_rot	: vector that contains the rotation value for each axis (rx,ry,rz).

getEulerAngles()

```
sl::float3 getEulerAngles (
            bool radian = true ) const
```

Convert the Rotation of the Transform as Euler angles.

Parameters

radian	: Define if the angle in is radian or degree

Returns

The Euler angles, as a sl::float3 (x=roll, y=pitch, z=yaw)

setEulerAngles()

```
void setEulerAngles (
            const sl::float3 & euler_angles,
            bool radian = true )
```

Sets the Rotation of the Transform from the Euler angles.

Parameters

euler_angles	: The Euler angles, as a sl::float3 (x=roll, y=pitch, z=yaw)
radian	: Define if the angle in is radian or degree

operator∗() [1/2]

```
Matrix4f operator* (
            const Matrix4f & mat ) const  [inherited]
```

brief Gives the result of the multiplication between two Matrix4f.

operator∗() [2/2]

```
Matrix4f operator* (
            const double & scalar ) const  [inherited]
```

brief Gives the result of the multiplication between a Matrix4f and a scalar.

operator==()

```
bool operator== (
            const Matrix4f & mat ) const  [inherited]
```

brief Test two Matrix4f equality.

operator"!=()

```
bool operator!= (
            const Matrix4f & mat ) const  [inherited]
```

brief Test two Matrix4f inequality.

operator()()

```
float& operator() (
            int u,
            int v )  [inherited]
```

Gets access to a specific point in the Matrix4f (read / write).

Parameters

u	: specify the row.
v	: specify the column.

Returns

The value at the u, v coordinates.

inverse() [1/2]

```
ERROR_CODE inverse ( )  [inherited]
```

Sets the Matrix4f to its inverse.

Returns

SUCCESS if the inverse has been computed, ERROR_CODE_FAILURE is not (det = 0).

inverse() [2/2]

```
static Matrix4f inverse (
            const Matrix4f & mat )  [static], [inherited]
```

Creates the inverse of a Matrix4f.

Parameters

rotation	: the Matrix4f to compute the inverse from.

Returns

The inverse of the given Matrix4f.

transpose() [1/2]

```
void transpose ( )  [inherited]
```

Sets the Matrix4f to its transpose.

transpose() [2/2]

```
static Matrix4f transpose (
            const Matrix4f & mat )  [static], [inherited]
```

Creates the transpose of a Matrix4f.

Parameters

rotation	: the Matrix4f to compute the transpose from.

Returns

The transpose of the given Matrix4f.

setIdentity()

```
void setIdentity ( )  [inherited]
```

Sets the Matrix4f to identity.

identity()

```
static Matrix4f identity ( )  [static], [inherited]
```

Creates an identity Matrix4f.

Returns

A Matrix4f set to identity.

setZeros()

```
void setZeros ( )  [inherited]
```

Sets the Matrix4f to zero.

zeros()

```
static Matrix4f zeros ( )  [static], [inherited]
```

Creates a Matrix4f filled with zeros.

Returns

A Matrix4f set to zero.

setSubMatrix3f()

```
ERROR_CODE setSubMatrix3f (
            sl::Matrix3f input,
            int row = 0,
            int column = 0 )  [inherited]
```

Sets a 3x3 Matrix inside the Matrix4f.

Note

Can be used to set the rotation matrix when the matrix4f is a pose or an isometric matrix.

Parameters

sl::Matrix3f	: sub matrix to put inside the Matrix4f.
row	: index of the row to start the 3x3 block. Must be 0 or 1.
column	: index of the column to start the 3x3 block. Must be 0 or 1.

Returns

SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

setSubVector3f()

```
ERROR_CODE setSubVector3f (
            sl::Vector3< float > input,
            int column = 3 )  [inherited]
```

Sets a 3x1 Vector inside the Matrix4f at the specified column index.

Note

Can be used to set the Translation/Position matrix when the matrix4f is a pose or an isometry.

Parameters

sl::Vector3	: sub vector to put inside the Matrix4f.
column	: index of the column to start the 3x3 block. By default, it is the last column (translation for a Pose).

Returns

SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

setSubVector4f()

```
ERROR_CODE setSubVector4f (
            sl::Vector4< float > input,
            int column = 3 )  [inherited]
```

Sets a 4x1 Vector inside the Matrix4f at the specified column index.

Note

Can be used to set the Translation/Position matrix when the matrix4f is a pose or an isometry.

Parameters

sl::Vector4	: sub vector to put inside the Matrix4f.
column	: index of the column to start the 3x3 block. By default, it is the last column (translation for a Pose).

Returns

SUCCESS if everything went well, ERROR_CODE_FAILURE otherwise.

getInfos()

```
sl::String getInfos ( )  [inherited]
```

Return the components of the Matrix4f in a sl::String.

Returns

A sl::String containing the components of the current Matrix4f.

Member Data Documentation

matrix_name

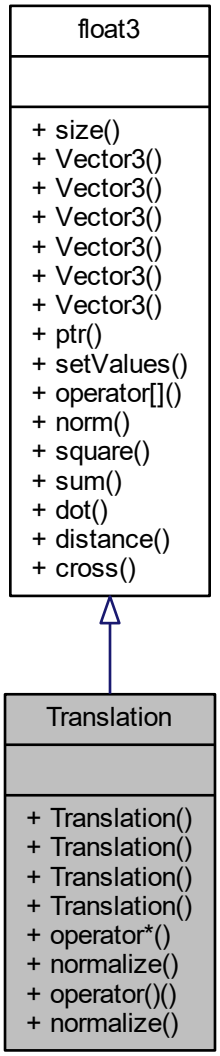
```
sl::String matrix_name  [inherited]
```

Name of the matrix (optional).

Translation Class Reference

Designed to contain translation data of the positional tracking.

Inheritance diagram for Translation:



Public Member Functions

- Translation ()

  empty Translation default constructor.
- Translation (const Translation &translation)

  Translation copy constructor (deep copy).
- Translation (float t1, float t2, float t3)

  Translation constructor.
- Translation (sl::float3 in)

  Translation constructor.
- Translation operator∗ (const Orientation &mat) const

  Multiplication operator by an Orientation.

- void normalize ()

  Normalizes the current translation.
- float & operator() (int x)

  Get the value at specific position in the Translation.
- _FCT_CPU_GPU_ int size () const
- _FCT_CPU_GPU_ const float * ptr () const
- _FCT_CPU_GPU_ Vector3< float > & setValues (const float *b)
- _FCT_CPU_GPU_ float & operator[ ] (int i)
- _FCT_CPU_GPU_ float norm ()

  returns the norm of the vector
- _FCT_CPU_GPU_ float square ()

  returns the squared norm of the vector
- _FCT_CPU_GPU_ float sum ()

  returns the sum of the vector

Static Public Member Functions

- static Translation normalize (const Translation &tr)

  Get the normalized version of a given Translation.
- static _FCT_CPU_GPU_ float dot (const Vector3< float > &a, const Vector3< float > &b)

  returns the dot product of two vector
- static _FCT_CPU_GPU_ float distance (const Vector3< float > &a, const Vector3< float > &b)

  returns the distance between two vector
- static _FCT_CPU_GPU_ Vector3< float > cross (const Vector3< float > &a, const Vector3< float > &b)

  returns the cross product between two vector

Detailed Description

Designed to contain translation data of the positional tracking.

sl::Translation is a vector as [tx, ty, tz]. You can access the data with the 't' ptr or by element name as : tx, ty, tz <-> | 0 1 2 |

Constructor & Destructor Documentation

Translation() [1/4]

```
Translation ( )
```

empty Translation default constructor.

Translation() [2/4]

```
Translation (
            const Translation & translation )
```

Translation copy constructor (deep copy).

Parameters

translation	: the Translation to copy.

Translation() [3/4]

```
Translation (
            float t1,
            float t2,
            float t3 )
```

Translation constructor.

Parameters

t1	: the x translation.
t2	: the y translation.
t3	: the z translation.

Translation() [4/4]

```
Translation (
            sl::float3 in )
```

Translation constructor.

Parameters

in	: vector.

Member Function Documentation

operator∗()

```
Translation operator* (
            const Orientation & mat ) const
```

Multiplication operator by an Orientation.

Parameters

mat	: Orientation.

Returns

The current Translation after being multiplied by the orientation.

normalize() [1/2]

```
void normalize ( )
```

Normalizes the current translation.

normalize() [2/2]

```
static Translation normalize (
            const Translation & tr )  [static]
```

Get the normalized version of a given Translation.

Parameters

tr	: the Translation to be used.

Returns

An other Translation object, which is equal to tr.normalize.

operator()()

```
float& operator() (
            int x )
```

Get the value at specific position in the Translation.

Parameters

x	: the position of the value

Returns

The value at the x position.

size()

```
_FCT_CPU_GPU_ int size ( ) const  [inline], [inherited]
```

ptr()

```
_FCT_CPU_GPU_ const float * ptr ( ) const  [inline], [inherited]
```

setValues()

```
_FCT_CPU_GPU_ Vector3<float >& setValues (
            const float * b )  [inline], [inherited]
```

operator[]()

```
_FCT_CPU_GPU_ float & operator[] (
            int i )  [inline], [inherited]
```

norm()

```
_FCT_CPU_GPU_ float norm ( )  [inline], [inherited]
```

returns the norm of the vector

square()

```
_FCT_CPU_GPU_ float square ( )  [inline], [inherited]
```

returns the squared norm of the vector

sum()

```
_FCT_CPU_GPU_ float sum ( )  [inline], [inherited]
```

returns the sum of the vector

dot()

```
static _FCT_CPU_GPU_ float dot (
            const Vector3< float > & a,
            const Vector3< float > & b )  [inline], [static], [inherited]
```

returns the dot product of two vector

distance()

```
static _FCT_CPU_GPU_ float distance (
            const Vector3< float > & a,
            const Vector3< float > & b )  [inline], [static], [inherited]
```

returns the distance between two vector

cross()

```
static _FCT_CPU_GPU_ Vector3<float > cross (
            const Vector3< float > & a,
            const Vector3< float > & b )  [inline], [static], [inherited]
```

returns the cross product between two vector

Vector2< T > Class Template Reference

Represents a two dimensions vector for both CPU and GPU.

Public Member Functions

- _FCT_CPU_GPU_ int size () const
- _FCT_CPU_GPU_ Vector2 ()
- _FCT_CPU_GPU_ Vector2 (const T &t)
- _FCT_CPU_GPU_ Vector2 (const T *tp)
- _FCT_CPU_GPU_ Vector2 (const T v0, const T v1)
- _FCT_CPU_GPU_ Vector2 (const Vector2< T > &v)
- _FCT_CPU_GPU_ const T * ptr () const
- _FCT_CPU_GPU_ Vector2 & setValues (const T *b)
- _FCT_CPU_GPU_ T & operator[ ] (int i)
- _FCT_CPU_GPU_ float norm ()

  returns the norm of the vector
- _FCT_CPU_GPU_ float square ()

  returns the squared norm of the vector
- _FCT_CPU_GPU_ float sum ()

  returns the sum of the vector
- _FCT_CPU_GPU_ float dot (const Vector2< T > &a, const Vector2< T > &b)

  returns the dot product of two vector
- _FCT_CPU_GPU_ float distance (const Vector2< T > &a, const Vector2< T > &b)

  returns the distance between two vector

Friends

- _FCT_CPU_GPU_ friend Vector2< T > & operator*= (Vector2< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector2< T > & operator*= (Vector2< T > &itself, const Vector2< T > &b)

- _FCT_CPU_GPU_ friend Vector2< T > & operator/= (Vector2< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector2< T > & operator/= (Vector2< T > &itself, const Vector2< T > &b)
- _FCT_CPU_GPU_ friend Vector2< T > & operator+= (Vector2< T > &itself, const Vector2< T > &b)
- _FCT_CPU_GPU_ friend Vector2< T > & operator-= (Vector2< T > &itself, const Vector2< T > &b)
- _FCT_CPU_GPU_ friend Vector2< T > operator+ (const Vector2< T > &a, const Vector2< T > &b)
- _FCT_CPU_GPU_ friend Vector2< T > operator- (const Vector2< T > &a, const Vector2< T > &b)
- _FCT_CPU_GPU_ friend Vector2< T > operator* (const Vector2< T > &a, T b)
- _FCT_CPU_GPU_ friend Vector2< T > operator* (const Vector2< T > &a, const Vector2< T > &b)
- _FCT_CPU_GPU_ friend Vector2< T > operator/ (const Vector2< T > &a, T b)
- _FCT_CPU_GPU_ friend Vector2< T > operator/ (const Vector2< T > &a, const Vector2< T > &b)

Detailed Description

template<typename T>
class sl::Vector2< T >

Represents a two dimensions vector for both CPU and GPU.

Constructor & Destructor Documentation

Vector2() [1/5]

```
_FCT_CPU_GPU_ Vector2 ( )  [inline]
```

Vector2() [2/5]

```
_FCT_CPU_GPU_ Vector2 (
            const T & t )  [inline]
```

Vector2() [3/5]

```
_FCT_CPU_GPU_ Vector2 (
            const T * tp )  [inline]
```

Vector2() [4/5]

```
_FCT_CPU_GPU_ Vector2 (
            const T v0,
            const T v1 )  [inline]
```

Vector2() [5/5]

```
_FCT_CPU_GPU_ Vector2 (
            const Vector2< T > & v )  [inline]
```

Member Function Documentation

size()

```
_FCT_CPU_GPU_ int size ( ) const  [inline]
```

ptr()

```
_FCT_CPU_GPU_ const T* ptr ( ) const  [inline]
```

setValues()

```
_FCT_CPU_GPU_ Vector2& setValues (
            const T * b )  [inline]
```

operator[]()

```
_FCT_CPU_GPU_ T& operator[] (
            int i )  [inline]
```

norm()

```
_FCT_CPU_GPU_ float norm ( )  [inline]
```

returns the norm of the vector

square()

```
_FCT_CPU_GPU_ float square ( )  [inline]
```

returns the squared norm of the vector

sum()

```
_FCT_CPU_GPU_ float sum ( )  [inline]
```

returns the sum of the vector

dot()

```
_FCT_CPU_GPU_ float dot (
            const Vector2< T > & a,
            const Vector2< T > & b )  [inline]
```

returns the dot product of two vector

distance()

```
_FCT_CPU_GPU_ float distance (
            const Vector2< T > & a,
            const Vector2< T > & b )  [inline]
```

returns the distance between two vector

Friends And Related Function Documentation

operator*= [1/2]

```
_FCT_CPU_GPU_ friend Vector2<T>& operator*= (
            Vector2< T > & itself,
            T d )  [friend]
```

operator*= [2/2]

```
_FCT_CPU_GPU_ friend Vector2<T>& operator*= (
            Vector2< T > & itself,
            const Vector2< T > & b )  [friend]
```

operator/= [1/2]

```
_FCT_CPU_GPU_ friend Vector2<T>& operator/= (
```

```
            Vector2< T > & itself,
            T d )  [friend]
```

operator/= [2/2]

```
_FCT_CPU_GPU_ friend Vector2<T>& operator/= (
            Vector2< T > & itself,
            const Vector2< T > & b )  [friend]
```

operator+=

```
_FCT_CPU_GPU_ friend Vector2<T>& operator+= (
            Vector2< T > & itself,
            const Vector2< T > & b )  [friend]
```

operator-=

```
_FCT_CPU_GPU_ friend Vector2<T>& operator-= (
            Vector2< T > & itself,
            const Vector2< T > & b )  [friend]
```

operator+

```
_FCT_CPU_GPU_ friend Vector2<T> operator+ (
            const Vector2< T > & a,
            const Vector2< T > & b )  [friend]
```

operator-

```
_FCT_CPU_GPU_ friend Vector2<T> operator- (
            const Vector2< T > & a,
            const Vector2< T > & b )  [friend]
```

operator* [1/2]

```
_FCT_CPU_GPU_ friend Vector2<T> operator* (
            const Vector2< T > & a,
            T b )  [friend]
```

operator* [2/2]

```
_FCT_CPU_GPU_ friend Vector2<T> operator* (
            const Vector2< T > & a,
            const Vector2< T > & b )  [friend]
```

operator/ [1/2]

```
_FCT_CPU_GPU_ friend Vector2<T> operator/ (
            const Vector2< T > & a,
            T b )  [friend]
```

operator/ [2/2]

```
_FCT_CPU_GPU_ friend Vector2<T> operator/ (
            const Vector2< T > & a,
            const Vector2< T > & b )  [friend]
```

Vector3< T > Class Template Reference

Represents a three dimensions vector for both CPU and GPU.

Public Member Functions

- _FCT_CPU_GPU_ int size () const
- _FCT_CPU_GPU_ Vector3 ()
- _FCT_CPU_GPU_ Vector3 (const T &t)
- _FCT_CPU_GPU_ Vector3 (const T *tp)
- _FCT_CPU_GPU_ Vector3 (const T v0, const T v1, const T v2)
- _FCT_CPU_GPU_ Vector3 (const Vector3< T > &v)
- _FCT_CPU_GPU_ Vector3 (const Vector2< T > &v, const T d=0)
- _FCT_CPU_GPU_ const T * ptr () const
- _FCT_CPU_GPU_ Vector3< T > & setValues (const T *b)
- _FCT_CPU_GPU_ T & operator[ ] (int i)
- _FCT_CPU_GPU_ float norm ()

  returns the norm of the vector
- _FCT_CPU_GPU_ float square ()

  returns the squared norm of the vector
- _FCT_CPU_GPU_ float sum ()

  returns the sum of the vector

Static Public Member Functions

- static _FCT_CPU_GPU_ float dot (const Vector3< T > &a, const Vector3< T > &b)

  returns the dot product of two vector
- static _FCT_CPU_GPU_ float distance (const Vector3< T > &a, const Vector3< T > &b)

  returns the distance between two vector
- static _FCT_CPU_GPU_ Vector3< T > cross (const Vector3< T > &a, const Vector3< T > &b)

  returns the cross product between two vector

Friends

- _FCT_CPU_GPU_ friend Vector3< T > & operator+= (Vector3< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector3< T > & operator+= (Vector3< T > &itself, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > & operator-= (Vector3< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector3< T > & operator-= (Vector3< T > &itself, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > & operator*= (Vector3< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector3< T > & operator*= (Vector3< T > &itself, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > & operator/= (Vector3< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector3< T > & operator/= (Vector3< T > &itself, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > operator+ (const Vector3< T > &a, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > operator- (const Vector3< T > &a, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > operator* (const Vector3< T > &a, T b)
- _FCT_CPU_GPU_ friend Vector3< T > operator* (T a, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > operator* (const Vector3< T > &a, const Vector3< T > &b)
- _FCT_CPU_GPU_ friend Vector3< T > operator/ (const Vector3< T > &a, T b)
- _FCT_CPU_GPU_ friend Vector3< T > operator/ (const Vector3< T > &a, const Vector3< T > &b)

Detailed Description

template<typename T>
class sl::Vector3< T >

Represents a three dimensions vector for both CPU and GPU.

Constructor & Destructor Documentation

Vector3() [1/6]

```
_FCT_CPU_GPU_ Vector3 ( )  [inline]
```

Vector3() [2/6]

```
_FCT_CPU_GPU_ Vector3 (
             const T & t )  [inline]
```

Vector3() [3/6]

```
_FCT_CPU_GPU_ Vector3 (
             const T * tp )  [inline]
```

Vector3() [4/6]

```
_FCT_CPU_GPU_ Vector3 (
             const T v0,
             const T v1,
             const T v2 )  [inline]
```

Vector3() [5/6]

```
_FCT_CPU_GPU_ Vector3 (
             const Vector3< T > & v )  [inline]
```

Vector3() [6/6]

```
_FCT_CPU_GPU_ Vector3 (
             const Vector2< T > & v,
             const T d = 0 )  [inline]
```

Member Function Documentation

size()

```
_FCT_CPU_GPU_ int size ( ) const  [inline]
```

ptr()

```
_FCT_CPU_GPU_ const T* ptr ( ) const  [inline]
```

setValues()

```
_FCT_CPU_GPU_ Vector3<T>& setValues (
             const T * b )  [inline]
```

operator[]()

```
_FCT_CPU_GPU_ T& operator[ ] (
            int i )  [inline]
```

norm()

```
_FCT_CPU_GPU_ float norm ( )  [inline]
```

returns the norm of the vector

square()

```
_FCT_CPU_GPU_ float square ( )  [inline]
```

returns the squared norm of the vector

sum()

```
_FCT_CPU_GPU_ float sum ( )  [inline]
```

returns the sum of the vector

dot()

```
static _FCT_CPU_GPU_ float dot (
            const Vector3< T > & a,
            const Vector3< T > & b )  [inline], [static]
```

returns the dot product of two vector

distance()

```
static _FCT_CPU_GPU_ float distance (
            const Vector3< T > & a,
            const Vector3< T > & b )  [inline], [static]
```

returns the distance between two vector

cross()

```
static _FCT_CPU_GPU_ Vector3<T> cross (
            const Vector3< T > & a,
            const Vector3< T > & b )  [inline], [static]
```

returns the cross product between two vector

Friends And Related Function Documentation

operator+= [1/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator+= (
            Vector3< T > & itself,
            T d )  [friend]
```

operator+= [2/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator+= (
            Vector3< T > & itself,
            const Vector3< T > & b )  [friend]
```

operator-= [1/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator-= (
            Vector3< T > & itself,
            T d )  [friend]
```

operator-= [2/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator-= (
            Vector3< T > & itself,
            const Vector3< T > & b )  [friend]
```

operator*= [1/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator*= (
            Vector3< T > & itself,
            T d )  [friend]
```

operator*= [2/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator*= (
            Vector3< T > & itself,
            const Vector3< T > & b )  [friend]
```

operator/= [1/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator/= (
            Vector3< T > & itself,
            T d )  [friend]
```

operator/= [2/2]

```
_FCT_CPU_GPU_ friend Vector3<T>& operator/= (
            Vector3< T > & itself,
            const Vector3< T > & b )  [friend]
```

operator+

```
_FCT_CPU_GPU_ friend Vector3<T> operator+ (
            const Vector3< T > & a,
            const Vector3< T > & b )  [friend]
```

operator-

```
_FCT_CPU_GPU_ friend Vector3<T> operator- (
            const Vector3< T > & a,
            const Vector3< T > & b )  [friend]
```

operator* [1/3]

```
_FCT_CPU_GPU_ friend Vector3<T> operator* (
            const Vector3< T > & a,
            T b )  [friend]
```

operator* [2/3]

```
_FCT_CPU_GPU_ friend Vector3<T> operator* (
            T a,
            const Vector3< T > & b )  [friend]
```

operator* [3/3]

```
_FCT_CPU_GPU_ friend Vector3<T> operator* (
            const Vector3< T > & a,
            const Vector3< T > & b )  [friend]
```

operator/ [1/2]

```
_FCT_CPU_GPU_ friend Vector3<T> operator/ (
            const Vector3< T > & a,
            T b )  [friend]
```

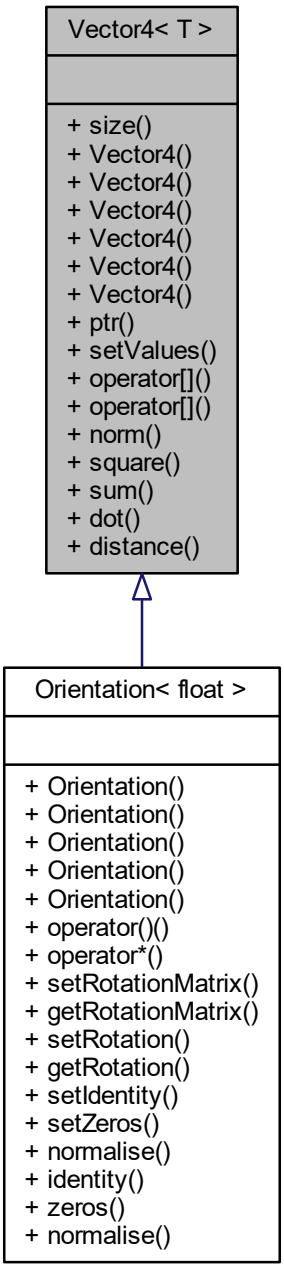
operator/ [2/2]

```
_FCT_CPU_GPU_ friend Vector3<T> operator/ (
            const Vector3< T > & a,
            const Vector3< T > & b )  [friend]
```

Vector4< T > Class Template Reference

Represents a four dimensions vector for both CPU and GPU.

Inheritance diagram for Vector4< T >:



Public Member Functions

- _FCT_CPU_GPU_ int size () const
- _FCT_CPU_GPU_ Vector4 ()
- _FCT_CPU_GPU_ Vector4 (const T &t)
- _FCT_CPU_GPU_ Vector4 (const T *tp)
- _FCT_CPU_GPU_ Vector4 (const T v0, const T v1, const T v2, const T v3)

- _FCT_CPU_GPU_ Vector4 (const Vector4< T > &v)
- _FCT_CPU_GPU_ Vector4 (const Vector4< T > &v, const T d)
- _FCT_CPU_GPU_ const T * ptr () const
- _FCT_CPU_GPU_ Vector4< T > & setValues (const T *b)
- _FCT_CPU_GPU_ T & operator[ ] (int i)
- _FCT_CPU_GPU_ const T & operator[ ] (int i) const
- _FCT_CPU_GPU_ float norm ()
  returns the norm of the vector
- _FCT_CPU_GPU_ float square ()
  returns the squared norm of the vector
- _FCT_CPU_GPU_ float sum ()
  returns the sum of the vector

Static Public Member Functions

- static _FCT_CPU_GPU_ float dot (const Vector4< T > &a, const Vector4< T > &b)
  returns the dot product of two vector
- static _FCT_CPU_GPU_ float distance (const Vector4< T > &a, const Vector4< T > &b)
  returns the distance between two vector

Friends

- _FCT_CPU_GPU_ friend Vector4< T > & operator+= (Vector4< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector4< T > & operator+= (Vector4< T > &itself, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > & operator-= (Vector4< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector4< T > & operator-= (Vector4< T > &itself, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > & operator*= (Vector4< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector4< T > & operator*= (Vector4< T > &itself, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > & operator/= (Vector4< T > &itself, T d)
- _FCT_CPU_GPU_ friend Vector4< T > & operator/= (Vector4< T > &itself, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > operator- (const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > operator+ (const Vector4< T > &a, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > operator- (const Vector4< T > &a, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > operator* (const Vector4< T > &a, T b)
- _FCT_CPU_GPU_ friend Vector4< T > operator* (T a, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > operator* (const Vector4< T > &a, const Vector4< T > &b)
- _FCT_CPU_GPU_ friend Vector4< T > operator/ (const Vector4< T > &a, T b)
- _FCT_CPU_GPU_ friend Vector4< T > operator/ (const Vector4< T > &a, const Vector4< T > &b)

Detailed Description

template<typename T>
class sl::Vector4< T >

Represents a four dimensions vector for both CPU and GPU.

Constructor & Destructor Documentation

Vector4() [1/6]

```
_FCT_CPU_GPU_ Vector4 ( )  [inline]
```

Vector4() [2/6]

```
_FCT_CPU_GPU_ Vector4 (
            const T & t )  [inline]
```

Vector4() [3/6]

```
_FCT_CPU_GPU_ Vector4 (
            const T * tp )  [inline]
```

Vector4() [4/6]

```
_FCT_CPU_GPU_ Vector4 (
            const T v0,
            const T v1,
            const T v2,
            const T v3 )  [inline]
```

Vector4() [5/6]

```
_FCT_CPU_GPU_ Vector4 (
            const Vector4< T > & v )  [inline]
```

Vector4() [6/6]

```
_FCT_CPU_GPU_ Vector4 (
            const Vector4< T > & v,
            const T d )  [inline]
```

Member Function Documentation

size()

```
_FCT_CPU_GPU_ int size ( ) const  [inline]
```

ptr()

```
_FCT_CPU_GPU_ const T* ptr ( ) const  [inline]
```

setValues()

```
_FCT_CPU_GPU_ Vector4<T>& setValues (
            const T * b )  [inline]
```

operator[]() [1/2]

```
_FCT_CPU_GPU_ T& operator[] (
            int i )  [inline]
```

operator[]() [2/2]

```
_FCT_CPU_GPU_ const T& operator[] (
            int i ) const  [inline]
```

norm()

```
_FCT_CPU_GPU_ float norm ( )  [inline]
```

returns the norm of the vector

square()

```
_FCT_CPU_GPU_ float square ( )  [inline]
```

returns the squared norm of the vector

sum()

```
_FCT_CPU_GPU_ float sum ( )  [inline]
```

returns the sum of the vector

dot()

```
static _FCT_CPU_GPU_ float dot (
            const Vector4< T > & a,
            const Vector4< T > & b )  [inline], [static]
```

returns the dot product of two vector

distance()

```
static _FCT_CPU_GPU_ float distance (
            const Vector4< T > & a,
            const Vector4< T > & b )  [inline], [static]
```

returns the distance between two vector

Friends And Related Function Documentation

operator+= [1/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator+= (
            Vector4< T > & itself,
            T d )  [friend]
```

operator+= [2/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator+= (
            Vector4< T > & itself,
            const Vector4< T > & b )  [friend]
```

operator-= [1/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator-= (
            Vector4< T > & itself,
            T d )  [friend]
```

operator-= [2/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator-= (
            Vector4< T > & itself,
            const Vector4< T > & b )  [friend]
```

operator*= [1/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator*= (
            Vector4< T > & itself,
            T d )  [friend]
```

operator*= [2/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator*= (
            Vector4< T > & itself,
            const Vector4< T > & b )  [friend]
```

operator/= [1/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator/= (
            Vector4< T > & itself,
            T d )  [friend]
```

operator/= [2/2]

```
_FCT_CPU_GPU_ friend Vector4<T>& operator/= (
            Vector4< T > & itself,
            const Vector4< T > & b )  [friend]
```

operator- [1/2]

```
_FCT_CPU_GPU_ friend Vector4<T> operator- (
            const Vector4< T > & b )  [friend]
```

operator+

```
_FCT_CPU_GPU_ friend Vector4<T> operator+ (
            const Vector4< T > & a,
            const Vector4< T > & b )  [friend]
```

operator- [2/2]

```
_FCT_CPU_GPU_ friend Vector4<T> operator- (
            const Vector4< T > & a,
            const Vector4< T > & b )  [friend]
```

operator* [1/3]

```
_FCT_CPU_GPU_ friend Vector4<T> operator* (
            const Vector4< T > & a,
            T b )  [friend]
```

operator* [2/3]

```
_FCT_CPU_GPU_ friend Vector4<T> operator* (
            T a,
            const Vector4< T > & b )  [friend]
```

operator* [3/3]

```
_FCT_CPU_GPU_ friend Vector4<T> operator* (
            const Vector4< T > & a,
            const Vector4< T > & b )  [friend]
```

operator/ [1/2]

```
_FCT_CPU_GPU_ friend Vector4<T> operator/ (
            const Vector4< T > & a,
            T b )  [friend]
```

operator/ [2/2]

```
_FCT_CPU_GPU_ friend Vector4<T> operator/ (
             const Vector4< T > & a,
             const Vector4< T > & b )  [friend]
```

Index