Python

Nov 17, 2018

Table of Contents

1 Example Outputs **3**
- 1.1 Machine Hall . . . 3
- 1.2 Cow and Lady Dataset . . . 4
- 1.3 Constrained Hardware . . . 4
- 1.4 KITTI Dataset . . . 4
- 1.5 EuRoC Dataset . . . 5
- 1.6 Beach Mapping . . . 5

2 Performance **7**
- 2.1 Integrator Performance . . . 7
- 2.2 Runtime Comparison to Octomap . . . 8

3 Installation **11**

4 Running Voxblox **13**

5 The Voxblox Node **15**
- 5.1 Table of Contents . . . 15
- 5.2 Published and Subscribed Topics . . . 15
  - 5.2.1 Published Topics . . . 16
  - 5.2.2 Subscribed Topics . . . 16
- 5.3 Services . . . 17
- 5.4 Parameters . . . 17
  - 5.4.1 General Parameters . . . 17
  - 5.4.2 TSDF Integrator Parameters . . . 17
  - 5.4.3 Fast TSDF Integrator Specific Parameters . . . 18
  - 5.4.4 ESDF Integrator Parameters . . . 18
  - 5.4.5 ICP Refinement Parameters . . . 19
  - 5.4.6 Input Transform Parameters . . . 19
  - 5.4.7 Output Parameters . . . 19

6 Using Voxblox for Planning **21**

7 How Does ESDF Generation Work? **25**
- 7.1 Description of the algorithm . . . 25
  - 7.1.1 How TSDF values are propagated to ESDF . . . 26
  - 7.1.2 How the Raise Wavefront works . . . 28

7.1.3 How the Lower Wavefront works . . . 30

8 Transformations in Voxblox **31**

9 Contributing to Voxblox **33**
9.1 Code style . . . 33
9.1.1 Setting up the linter . . . 33
9.2 Modifying Voxblox . . . 33
9.2.1 Serialization . . . 34

10 Library API **37**
10.1 Class Hierarchy . . . 37
10.2 File Hierarchy . . . 37
10.3 Full API . . . 37
10.3.1 Namespaces . . . 37
10.3.2 Classes and Structs . . . 49
10.3.3 Enums . . . 116
10.3.4 Functions . . . 118
10.3.5 Variables . . . 150
10.3.6 Typedefs . . . 152
10.3.7 Directories . . . 160
10.3.8 Files . . . 163

11 Paper and Video **391**

12 Credits **393**

Voxblox is a volumetric mapping library based mainly on Truncated Signed Distance Fields (TSDFs). It varies from other SDF libraries in the following ways:

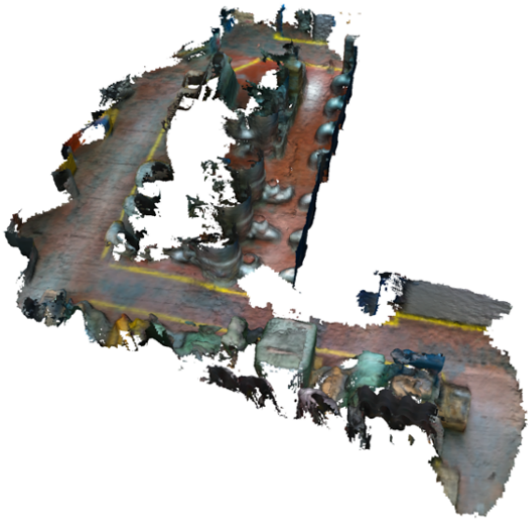
- CPU-only, can be run single-threaded or multi-threaded for some integrators
- Support for multiple different layer types (containing different types of voxels)
- Serialization using protobufs
- Different ways of handling weighting during merging
- Different ways of inserting pose information about scans
- Tight ROS integration (in voxblox_ros package)
- Easily extensible with whatever integrators you want
- Features an implementation of building Euclidean Signed Distance Fields (ESDFs, EDTs) directly from TSDFs.

CHAPTER 1

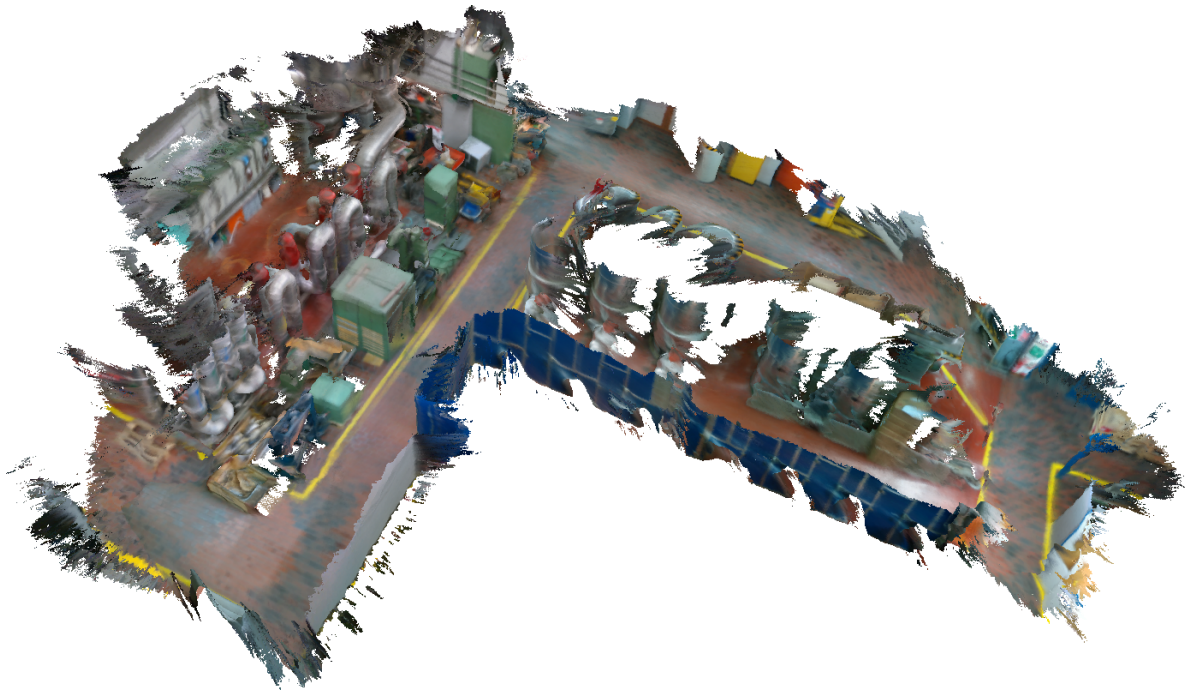
Example Outputs

1.1 Machine Hall

A mesh produced by Voxblox running inside a manifold mapper that fuses a SLAM systems poses with the output of a realsense D415 depthcamera. The map was generated while all systems were running fully onboard the pictured micro aerial vehicle.



A higher resolution mesh of the same area that was processed by voxblox offline is shown below.



1.2 Cow and Lady Dataset

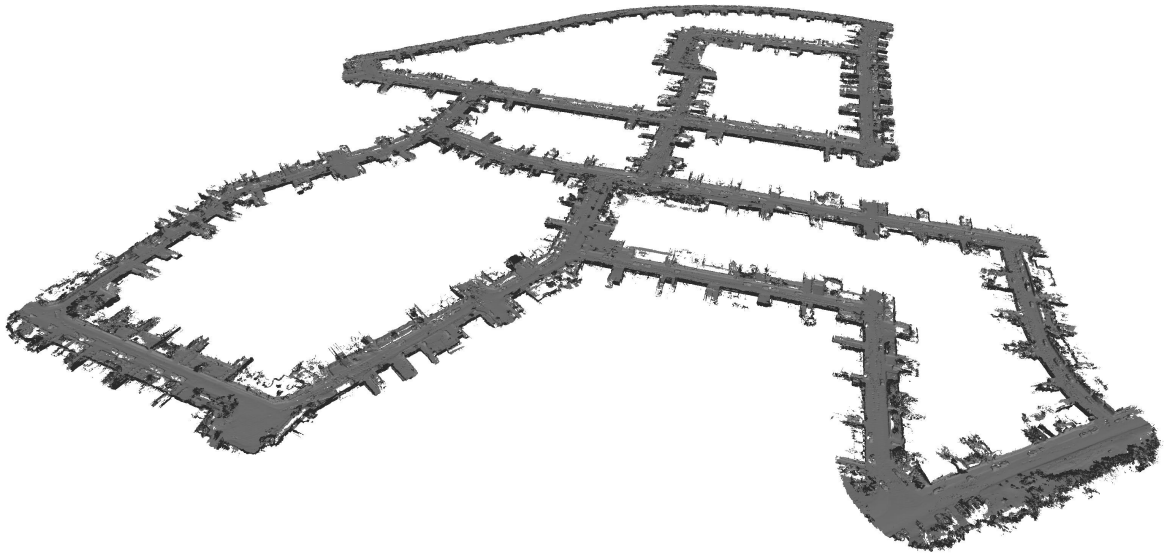
Voxblox running on the cow and lady dataset on a laptop equiped with an i7-4810MQ 2.80GHz CPU. In this example the system is integrating a TSDF, generating a mesh and publishing the result to RViz in real time.

1.3 Constrained Hardware

Voxblox running fully onboard the Atom processor of an Intel-Euclid. Again, the system is integrating, meshing and publishing in realtime. In this example the system was also sharing the CPU with the localization system (ROVIO) and the sensor drivers. This left around one CPU core for Voxblox to use.

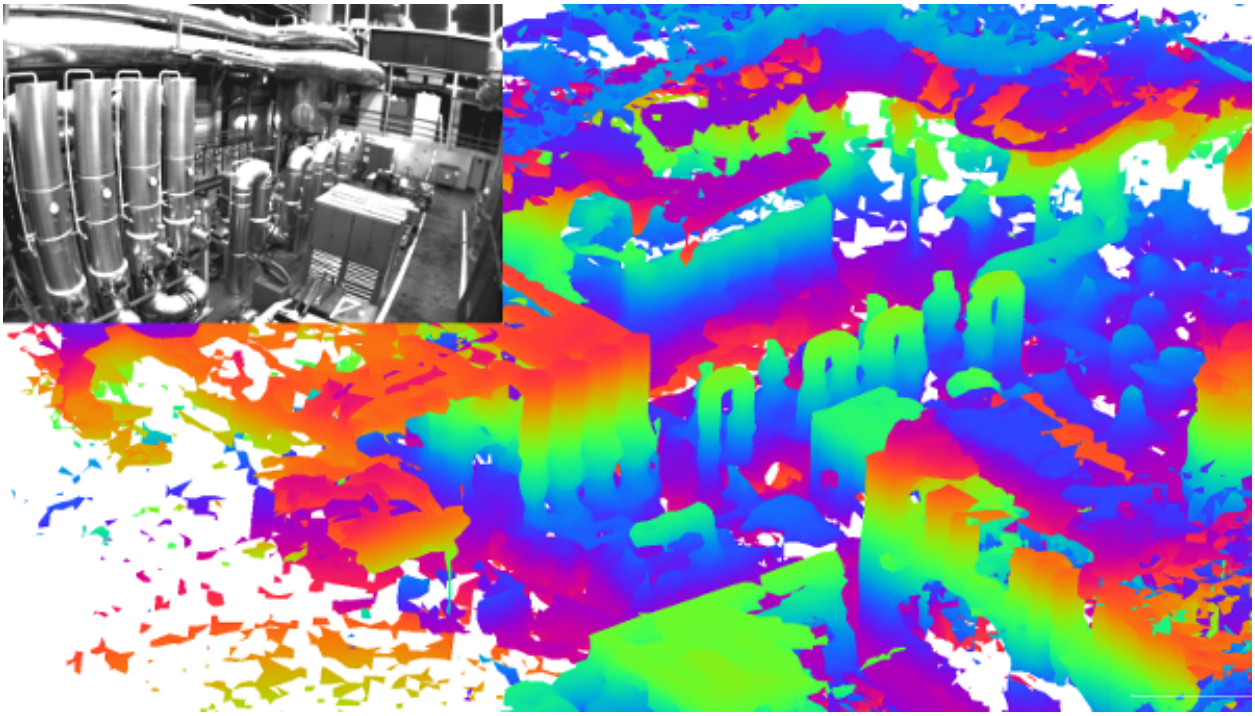
1.4 KITTI Dataset

A mesh produced from Voxblox when run on the KITTI dataset on a Desktop PC. The given localization solution and the pointcloud produced by the Velodyne were used.



1.5 EuRoC Dataset

A voxblox mesh produced by the Maplab library running on the Stereo data provided by the EuRoC dataset.



1.6 Beach Mapping

A map of a beach produced by a platform with two sets of stereo cameras flying an automated coverage path.

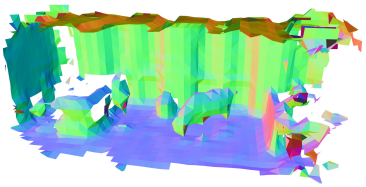
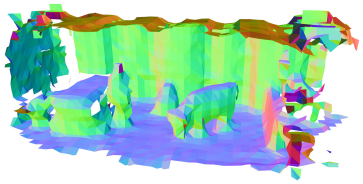
CHAPTER 2

Performance

The Voxblox code has prioritized readability and easy extension over performance. It was also designed to operate on systems that lack a GPU. One of the main drives to create Voxblox was to create a volumetric mapping library that fit the needs of planning for robots, because of this, and unlike many TSDF libraries all possible freespace is mapped in addition to areas close to surfaces. These design decisions limit performance, however high quality real-time mapping of large enviroments is still easily acheivable.

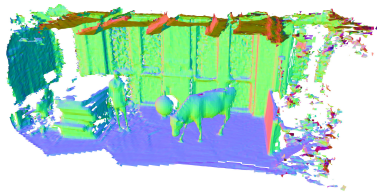
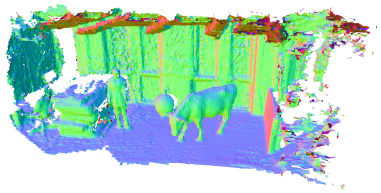
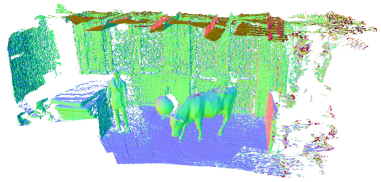
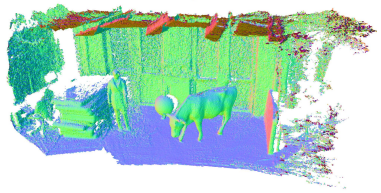
2.1 Integrator Performance

A table of demonstrating the performance of the merged and fast integrators on the cow and lady dataset on a i7-4810MQ 2.80GHz CPU is shown below:

Rendered Mesh	Setup	
	Integrator Voxel size TSDF Meshing Total RAM	Merged 20 cm 56 ms / scan 2 ms / scan 49 MB
	Integrator Voxel size TSDF Meshing Total RAM	Fast 20 cm 20 ms / scan 2 ms / scan 62 MB

Continued on next page

Table 1 – continued from previous page

Rendered Mesh	Setup	
	Integrator Voxel size TSDF Meshing Total RAM	Merged 5 cm 112 ms / scan 10 ms / scan 144 MB
	Integrator Voxel size TSDF Meshing Total RAM	Fast 5 cm 23 ms / scan 12 ms / scan 153 MB
	Integrator Voxel size TSDF Meshing Total RAM	Merged 2 cm 527 ms / scan 66 ms / scan 609 MB
	Integrator Voxel size TSDF Meshing Total RAM	Fast 2 cm 63 ms / scan 110 ms / scan 673 MB

2.2 Runtime Comparison to Octomap

Octomap ray casts all points from the origin, without bundling or any other approximation techniques. This is the same approach taken by voxblox's simple integrator. This leads to a significant difference in the integration times

when compared to voxblox's fast integrator. A comparison in the performance was run integrating the velodyne data from the first 60 seconds of the 2011_10_03_drive_0027_sync KITTI dataset. The test was performed on an Intel i7-4810MQ quad core CPU running at 2.8 GHz and truncation distance was set to 4 voxels. The results are shown in the table below:

Voxel size	Max ray length	Octomap integration time	Voxblox integration time
0.5 m	10 m	38 ms / scan	14 ms / scan
0.5 m	50 m	63 ms / scan	14 ms / scan
0.2 m	10 m	136 ms / scan	18 ms / scan
0.2 m	50 m	760 ms / scan	44 ms / scan
0.1 m	10 m	905 ms / scan	35 ms / scan
0.1 m	50 m	3748 ms / scan	100 ms / scan

On the same dataset voxblox was found to use 2 to 3 times the ram of Octomap.

CHAPTER 3

Installation

To install voxblox, please install ROS Indigo, ROS Kinetic or ROS Melodic. These instructions are for Ubuntu, Voxblox will also run on OS X, but you're more or less on your own there.

First install additional system dependencies (swap kinetic for indigo or melodic as necessary):

```
sudo apt-get install python-wstool python-catkin-tools ros-kinetic-cmake-modules␣
↪protobuf-compiler autoconf
```

Next, add a few other dependencies. If you don't have a catkin workspace yet, set it up as follows:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws
catkin init
catkin config --extend /opt/ros/kinetic
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release
catkin config --merge-devel
```

If using SSH keys for github (recommended):

```
cd ~/catkin_ws/src/
git clone git@github.com:ethz-asl/voxblox.git
wstool init . ./voxblox/voxblox_ssh.rosinstall
wstool update
```

If **not using SSH** keys but using https instead:

```
cd ~/catkin_ws/src/
git clone https://github.com/ethz-asl/voxblox.git
wstool init . ./voxblox/voxblox_https.rosinstall
wstool update
```

If you have already initalized wstool replace the above `wstool init` with `wstool merge -t`

Compile:

```
cd ~/catkin_ws/src/
catkin build voxblox_ros
```

If you wish to compile the online docs locally (not needed unless you wish to play with how your doxygen comments are rendered) you will need the following additional dependencies:

```
sudo apt-get install python-pip doxygen
pip install sphinx exhale breath sphinx_rtd_theme --user
```

The html documentation can then be compiled by:

```
cd ~/catkin_ws/src/voxblox/voxblox/docs
make html
```

CHAPTER 4

Running Voxblox

The easiest way to test out voxblox is to try it out on a dataset. We have launch files for our own dataset, the Euroc Vicon Room datasets, and the KITTI raw datasets processed through kitti_to_rosbag.

For each of these datasets, there's a launch file associated under *voxblox_ros/launch*.

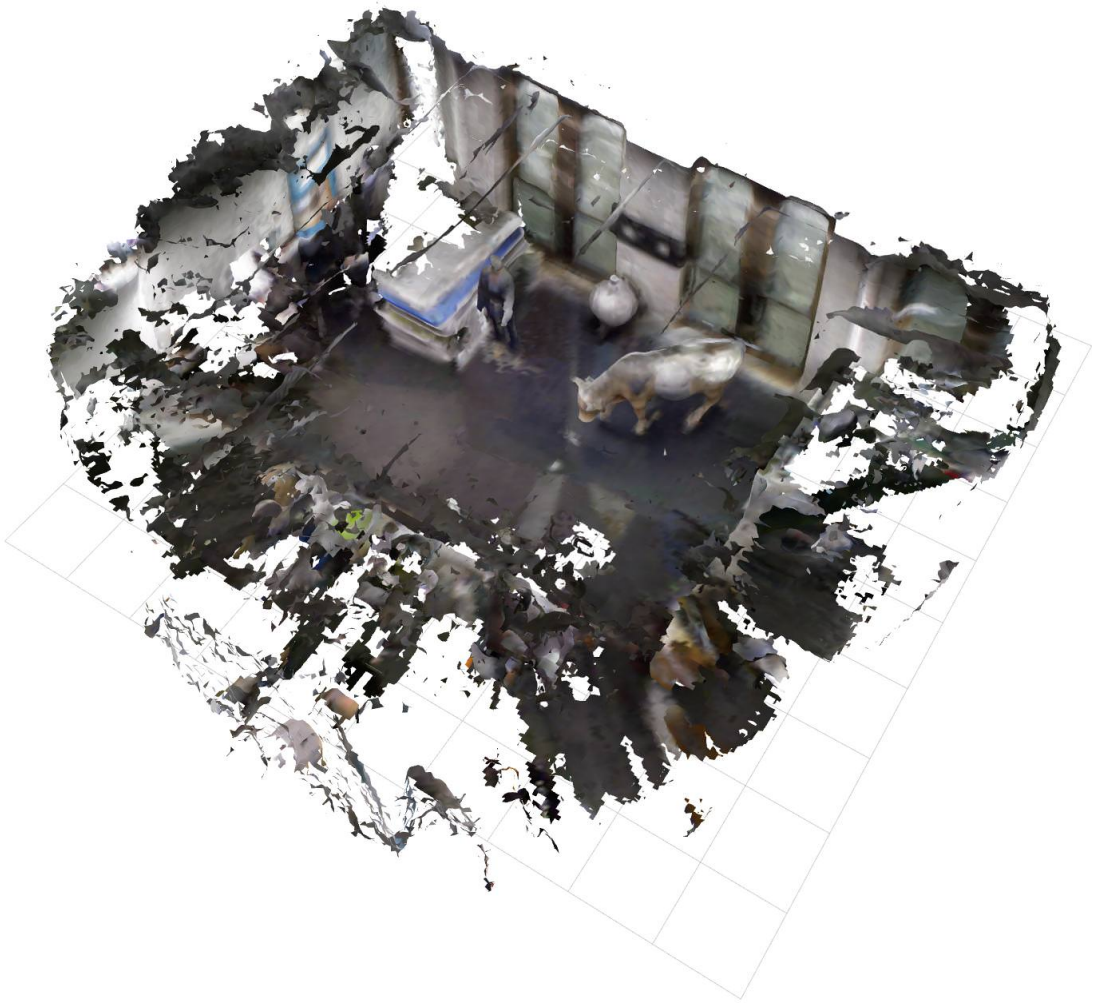
The easiest way to start is to download the cow and lady dataset, edit the path to the bagfile in `cow_and_lady_dataset.launch`, and then simply:

```
roslaunch voxblox_ros cow_and_lady_dataset.launch
```

An alternative dataset the basement dataset is also available. While this dataset lacks ground truth it demonstrates the capabilities of Voxblox running on Velodyne lidar data and uses ICP corrections to compensate for a drifting pose estimate. To run the dataset edit the path to the bagfile in `basement_dataset.launch`, and then simply:

```
roslaunch voxblox_ros basement_dataset.launch
```

If you open rviz, you should be able to see the the mesh visualized on the `/voxblox_node/mesh` MarkerArray topic, in the `world` static frame, as shown below. The mesh only updates once per second (this is a setting in the launch file).



The rest of the commonly-used settings are parameters in the launch file.

CHAPTER 5

The Voxblox Node

5.1 Table of Contents

- *Published and Subscribed Topics*
  - *Published Topics*
  - *Subscribed Topics*
- *Services*
- *Parameters*
  - *General Parameters*
  - *TSDF Integrator Parameters*
  - *Fast TSDF Integrator Specific Parameters*
  - *ESDF Integrator Parameters*
  - *ICP Refinement Parameters*
  - *Input Transform Parameters*
  - *Output Parameters*

5.2 Published and Subscribed Topics

Note: the voxblox_node has been replaced with tsdf_server (if you want a TSDF) or esdf_server (if you want both a TSDF and an ESDF). The tsdf_server and esdf_server publish and subscribe to the following topics:

5.2.1 Published Topics

mesh **voxblox_msgs::MeshBlock** A visualization topic showing the mesh produced from the tsdf in a form that can be seen in RViz. Set `update_mesh_every_n_sec` to control its update rate.

surface_pointcloud **pcl::PointCloud<pcl::PointXYZRGB>** A colored pointcloud of the voxels that are close to a surface.

tsdf_pointcloud **pcl::PointCloud<pcl::PointXYZI>** A pointcloud showing all allocated voxels.

mesh_pointcloud **pcl::PointCloud<pcl::PointXYZRGB>** Only appears if `output_mesh_as_pointcloud` is true, outputs a pointcloud containing the verticies of the generated mesh.

mesh_pcl **pcl_msgs::PolygonMesh** Only appears if `output_mesh_as_pcl_mesh` is true, outputs any mesh generated by the generate_mesh service.

tsdf_slice **pcl::PointCloud<pcl::PointXYZI>** Outputs a 2D horizontal slice of the TSDF colored by the stored distance value.

esdf_pointcloud **pcl::PointCloud<pcl::PointXYZI>** A pointcloud showing the values of all allocated ESDF voxels. Only appears if using `esdf_server`.

esdf_slice **pcl::PointCloud<pcl::PointXYZI>** Outputs a 2D horizontal slice of the ESDF colored by the stored distance value. Only appears if using `esdf_server`.

occupied_nodes **visualization_msgs::MarkerArray** Visualizes the location of the allocated voxels in the TSDF.

tsdf_map_out **voxblox_msgs::Layer** Publishes the entire TSDF layer to update other nodes (that listen on tsdf_layer_in). Only published if `publish_tsdf_map` is set to true.

esdf_map_out **voxblox_msgs::Layer** Publishes the entire ESDF layer to update other nodes (that listen on esdf_layer_in). Only published if `publish_esdf_map` is set to true.

traversable **pcl::PointCloud<pcl::PointXYZI>** (ESDF server only) Outputs all the points within the map that are considered traversable, controlled by the `publish_traversable` and `traversability_radius` parameters.

5.2.2 Subscribed Topics

transform **geometry_msgs::TransformStamped** Only appears if `use_tf_transforms` is false. The transformation from the world frame to the current sensor frame.

pointcloud **sensor_msgs::PointCloud2** The input pointcloud to be integrated.

freespace_pointcloud **sensor_msgs::PointCLoud2** Only appears if `use_freespace_pointcloud` is true. Unlike the `pointcloud` topic where the given points lie on surfaces, the points in the `freespace_pointcloud` are taken to be floating in empty space. These points can assist in generating more complete freespace information in a map.

tsdf_map_in **voxblox_msgs::Layer** Replaces the current TSDF layer with that from this topic. Voxel size and voxels per side should match.

esdf_map_in **voxblox_msgs::Layer** Replaces the current ESDF layer with that from this topic. Voxel size and voxels per side should match.

icp_transform **geometry_msgs::TransformStamped** If ICP is enabled, this is the current corrected transform between the world frame and the ICP frame.

5.3 Services

The tsdf_server and esdf_server have the following services:

generate_mesh This service has an empty request and response. Calling this service will generate a new mesh. The mesh will be saved as a ply file unless `mesh_filename` is set to "". The mesh will also be output on the `mesh_pointcloud` topic if `output_mesh_as_pointcloud` is true and on the `mesh_pcl` topic if `output_mesh_as_pcl_mesh` is true.

generate_esdf This service has an empty request and response. It can be used to trigger an esdf map update.

save_map This service has a `voxblox_msgs::FilePath::Request` and `voxblox_msgs::FilePath::Response`. The service call saves the tsdf layer to a .vxblx file.

load_map This service has a `voxblox_msgs::FilePath::Request` and `voxblox_msgs::FilePath::Response`. The service call loads the tsdf layer from a .vxblx file.

publish_map This service has an empty request and response. Publishes any TSDF and ESDF layers on the `tsdf_map_out` and `esdf_map_out` topics.

publish_pointclouds This service has an empty request and response. Publishes TSDF and ESDF pointclouds and slices.

5.4 Parameters

A summary of the user setable tsdf_server and esdf_server parameters. All parameters are listed as:

`Parameter` ***Default*** Description.

5.4.1 General Parameters

`min_time_between_msgs_sec` ***0.0*** Minimum time to wait after integrating a message before accepting a new one.

`pointcloud_queue_size` ***1*** The size of the queue used to subscribe to pointclouds.

`verbose` ***true*** Prints additional debug and timing information.

`max_block_distance_from_body` ***3.40282e+38*** Blocks that are more than this distance from the latest robot pose are deleted, saving memory.

5.4.2 TSDF Integrator Parameters

`method` *"merged"*

"simple" The most straightfoward integrator. Every point in the pointcloud has a ray cast from the origin through it. Every voxel each ray passes through is updated individually. A very slow and exact approach.

"merged" Rays that start and finish in the same voxel are bundled into a single ray. The properties of the points are merged and their weights added so no information is lost. The approximation means some voxels will recive updates that were otherwise meant for neighboring voxels. This approach works well with large voxels (10 cm or greater) and can give an order of magnitude speed up over the simple integrator.

"fast" Rays that attempt to update voxels already updated by other rays from the same pointcloud are terminated early and discarded. An approximate method that has been designed to give the fastest possible results at the expense of discarding large quantities of information. The trade off between speed and information loss can be tuned via the `start_voxel_subsampling_factor` and `max_consecutive_ray_collisions` parameters. This method is currently the only viable integrator for real-time applications with voxels smaller than 5 cm.

`tsdf_voxel_size` ***0.2*** The size of the tsdf voxels

`tsdf_voxels_per_side` ***16*** TSDF voxels per side of an allocated block. Must be a power of 2

`voxel_carving_enabled` ***true*** If true, the entire length of a ray is integrated, if false only the region inside the trunaction distance is used.

`truncation_distance` ***2*‘‘‘tsdf_voxel_size‘*** The truncation distance for the TSDF

`max_ray_length_m` ***5.0*** The maximum range out to which a ray will be cast

`min_ray_length_m` ***0.1*** The point at which the ray casting will start

`max_weight` ***10000.0*** The upper limit for the weight assigned to a voxel

`use_const_weight` ***false*** If true all points along a ray have equal weighting

`allow_clear` ***true*** If true points beyond the `max_ray_length_m` will be integrated up to this distance

`use_freespace_pointcloud` ***false*** If true a second subscription topic `freespace_pointcloud` appears. Clearing rays are cast from beyond this topic's points' truncation distance to assist in clearing freespace voxels

5.4.3 Fast TSDF Integrator Specific Parameters

These parameters are only used if the integrator `method` is set to "fast".

`start_voxel_subsampling_factor` ***2*** Before integration points are inserted into a sub-voxel, only one point is allowed per sub-voxel. This can be thought of as subsampling the pointcloud. The edge length of the sub-voxel is the voxel edge length divided by `start_voxel_subsampling_factor`.

`max_consecutive_ray_collisions` ***2*** When a ray is cast by this integrator it detects if any other ray has already passed through the current voxel this scan. If it passes through more than `max_consecutive_ray_collisions` voxels other rays have seen in a row, it is taken to be adding no new information and the casting stops.

`max_integration_time_s` ***3.40282e+38*** The time budget for frame integration, if this time is exceeded ray casting is stopped early. Used to guarantee real time performance.

`clear_checks_every_n_frames` ***1*** Governs how often the sets that indicate if a sub-voxel is full or a voxel has had a ray passed through it are cleared.

5.4.4 ESDF Integrator Parameters

`esdf_max_distance_m` ***2.0*** The maximum distance that the esdf will be calculated out to.

`esdf_default_distance_m` ***2.0*** Default distance set for unknown values and values >‘‘esdf_max_distance_m‘‘.

`clear_sphere_for_planning` ***false*** Enables setting unknown space to free near the current pose of the sensor, and unknown space to occupied further away from the sensor. Controlled by the two parameters below.

`clear_sphere_radius` ***1.5*** Radius of the inner sphere where unknown is set to free, in meters.

`occupied_sphere_radius` ***5.0*** Radius of the outer sphere where unknown is set to occupied, in meters.

5.4.5 ICP Refinement Parameters

ICP based refinement can be applied to the poses of the input pointclouds before merging.

enable_icp *false* Whether to use ICP to align all incoming pointclouds to the existing structure.

icp_refine_roll_pitch *true* True to apply 6-dof pose correction, false for 4-dof (x, y, z, yaw) correction.

accumulate_icp_corrections *true* Whether to accumulate transform corrections from ICP over all pointclouds. Reset at each new pointcloud if false.

icp_corrected_frame *icp_corrected* TF frame to output the ICP corrections to.

pose_corrected_frame *pose_corrected* TF frame used to output the ICP corrected poses relative to the `icp_corrected_frame`.

icp_mini_batch_size *20* Number of points used in each batch of point matching corrections.

icp_subsample_keep_ratio *0.5* Random subsampling will be used to reduce the number of points used for matching.

icp_min_match_ratio *0.8* For a mini batch refinement to be accepted, at least this ratio of points in the pointcloud must fall within the truncation distance of the existing TSDF layer.

icp_inital_translation_weighting *100.0* A rough measure of the confidence the system has in the provided inital pose. Each point used in ICP contributes 1 point of weighting information to the translation.

icp_inital_rotation_weighting *100.0* A rough measure of the confidence the system has in the provided inital pose. Each point used in ICP contributes 2 points of weighting information to the rotation.

5.4.6 Input Transform Parameters

use_tf_transforms *true* If true the ros tf tree will be used to get the pose of the sensor relative to the world (`sensor_frame` and `world_frame` will be used). If false the pose must be given via the `transform` topic.

world_frame *"world"* The base frame used when looking up tf transforms. This is also the frame that most outputs are given in.

sensor_frame *""* The sensor frame used when looking up tf transforms. If set to "" the frame of the input pointcloud message will be used.

T_B_D A static transformation from the base to the dynamic system that will be applied.

invert_T_B_D *false* If the given `T_B_D` should be inverted before it is used.

T_B_C A static transformation from the base to the sensor that will be applied.

invert_T_B_C *false* If the given `T_B_C` should be inverted before it is used.

5.4.7 Output Parameters

output_mesh_as_pointcloud *false* If true the verticies of the generated mesh will be ouput as a pointcloud on the topic `mesh_pointcloud` whenever the generate_mesh service is called.

output_mesh_as_pcl_mesh *false* If true the generated mesh will be ouput as a `pcl::PolygonMesh` on the topic `mesh_pcl` whenever the generate_mesh service is called.

slice_level *0.5* The height at which generated tsdf and esdf slices will be made.

color_ptcloud_by_weight *false* If the pointcloud should be colored by the voxel weighting.

mesh_filename *""* Filename output mesh will be saved to, leave blank if no file should be generated.

color_mode *"color"* The method that will be used for coloring the mesh. Options are "color", "height", "normals", "lambert" and "gray".

mesh_min_weight *1e-4* The minimum weighting needed for a point to be included in the mesh.

update_mesh_every_n_sec *0.0* Rate at which the mesh topic will be published to, a value of 0 disables. Note, this will not trigger any other mesh operations, such as generating a ply file.

publish_tsdf_map *false* Whether to publish the complete TSDF map periodically over ROS topics.

publish_esdf_map *false* Whether to publish the complete ESDF map periodically over ROS topics.

publish_tsdf_info *false* Enables publishing of `tsdf_pointcloud`, `surface_pointcloud` and `occupied_nodes`.

`publish_pointclouds` | If true the tsdf and esdf (if generated) is published as a pointcloud when the mesh is updated. `intensity_colormap` *"rainbow"*

If the incoming pointcloud is an intensity (not RGB) pointcloud, such as from laser, this sets how the intensities will be mapped to a color. Valid options are `rainbow`, `inverse_rainbow`, `grayscale`, `inverse_grayscale`, `ironbow` (thermal).

intensity_max_value *100.0* Maximum value to use for the intensity mapping. Minimum value is always 0.

publish_traversable *false* Whether to display a traversability pointcloud from an ESDF server.

traversability_radius *1.0* The minimum radius at which a point is considered traversable.

CHAPTER 6

Using Voxblox for Planning

The planners described in Continuous-Time Trajectory Optimization for Online UAV Replanning, Safe Local Exploration for Replanning in Cluttered Unknown Environments for Micro-Aerial Vehicles, and Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning will be open-sourced shortly.

In the mean-time, the general idea behind using voxblox for planning is to have two nodes running: one for the mapping, which ingests pointcloud data and produces both a TSDF and an ESDF, and one for planning, which subscribes to the latest ESDF layer over ROS.

The planner should have a `voxblox::EsdfServer` as a member, and simply remap the `esdf_map_out` and `esdf_map_in` topics to match.

A sample launch file is shown below:

```
<launch>
  <arg name="robot_name" default="my_robot" />
  <arg name="voxel_size" default="0.20" />
  <arg name="voxels_per_side" default="16" />
  <arg name="world_frame" default="odom" />
  <group ns="$(arg robot_name)">

    <node name="voxblox_node" pkg="voxblox_ros" type="esdf_server" output="screen"␣
↪args="-alsologtostderr" clear_params="true">
      <remap from="pointcloud" to="great_sensor/my_pointcloud"/>
      <remap from="voxblox_node/esdf_map_out" to="esdf_map" />
      <param name="tsdf_voxel_size" value="$(arg voxel_size)" />
      <param name="tsdf_voxels_per_side" value="$(arg voxels_per_side)" />
      <param name="publish_esdf_map" value="true" />
      <param name="publish_pointclouds" value="true" />
      <param name="use_tf_transforms" value="true" />
      <param name="update_mesh_every_n_sec" value="1.0" />
      <param name="clear_sphere_for_planning" value="true" />
      <param name="world_frame" value="$(arg world_frame)" />
    </node>

    <node name="my_voxblox_planner" pkg="voxblox_planner" type="my_voxblox_planner"␣
↪output="screen" args="-alsologtostderr">
```

(continues on next page)

(continued from previous page)

```
      <remap from="odometry" to="great_estimator/odometry" />
      <remap from="my_voxblox_planner/esdf_map_in" to="esdf_map" />
      <param name="tsdf_voxel_size" value="$(arg voxel_size)" />
      <param name="tsdf_voxels_per_side" value="$(arg voxels_per_side)" />
      <param name="update_mesh_every_n_sec" value="0.0" />
      <param name="world_frame" value="$(arg world_frame)" />
    </node>

  </group>
</launch>
```

And some scaffolding for writing your own planner using ESDF collision checking:

```
class YourPlannerVoxblox {
 public:
  YourPlannerVoxblox(const ros::NodeHandle& nh,
                     const ros::NodeHandle& nh_private);
  virtual ~YourPlannerVoxblox() {}
  double getMapDistance(const Eigen::Vector3d& position) const;
 private:
  ros::NodeHandle nh_;
  ros::NodeHandle nh_private_;

  // Map!
  voxblox::EsdfServer voxblox_server_;
};
```

There's also a traversability pointcloud you can enable/disable, that if you set the radius to your robot's collision checking radius, can show you parts of the map the planner thinks are traversable in a pointcloud:

```
YourPlannerVoxblox::YourPlannerVoxblox(const ros::NodeHandle& nh,
                                       const ros::NodeHandle& nh_private)
    : nh_(nh),
      nh_private_(nh_private),
      voxblox_server_(nh_, nh_private_) {
  // Optionally load a map saved with the save_map service call in voxblox.
  std::string input_filepath;
  nh_private_.param("voxblox_path", input_filepath, input_filepath);
  if (!input_filepath.empty()) {
    if (!voxblox_server_.loadMap(input_filepath)) {
      ROS_ERROR("Couldn't load ESDF map!");
    }
  }
  double robot_radius = 1.0;
  voxblox_server_.setTraversabilityRadius(robot_radius);
  voxblox_server_.publishTraversable();
}
```

Then to check for collisions you can just compare map distance to your robot radius:

```
double YourPlannerVoxblox::getMapDistance(
    const Eigen::Vector3d& position) const {
  if (!voxblox_server_.getEsdfMapPtr()) {
    return 0.0;
  }
  double distance = 0.0;
```

(continues on next page)

(continued from previous page)

```
  if (!voxblox_server_.getEsdfMapPtr()->getDistanceAtPosition(position,
                                                              &distance)) {
    return 0.0;
  }
  return distance;
}
```

CHAPTER 7

How Does ESDF Generation Work?

7.1 Description of the algorithm

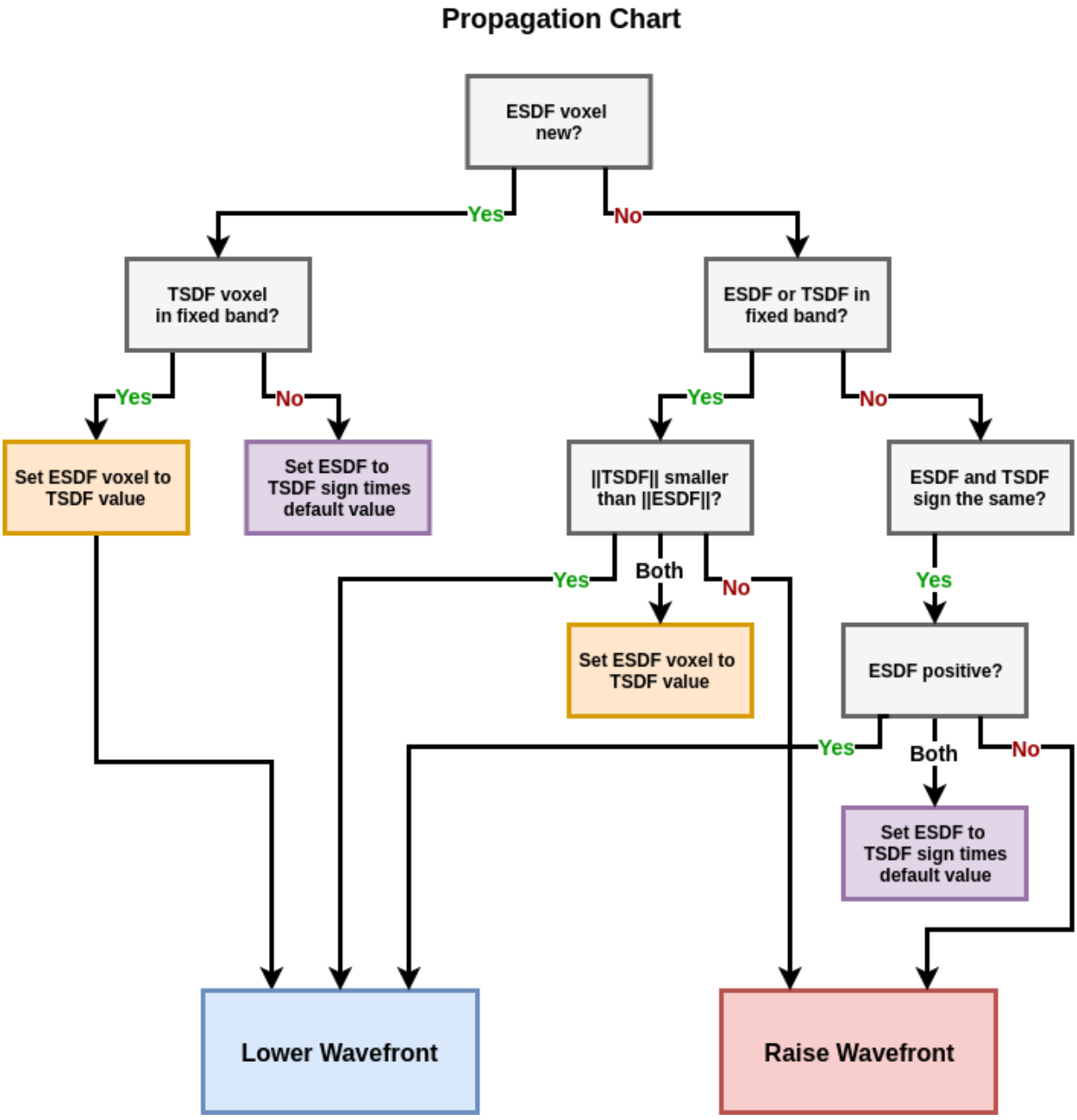
The algorithm id described in this paper:

Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan Nieto, and Roland Siegwart, "**Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning**", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

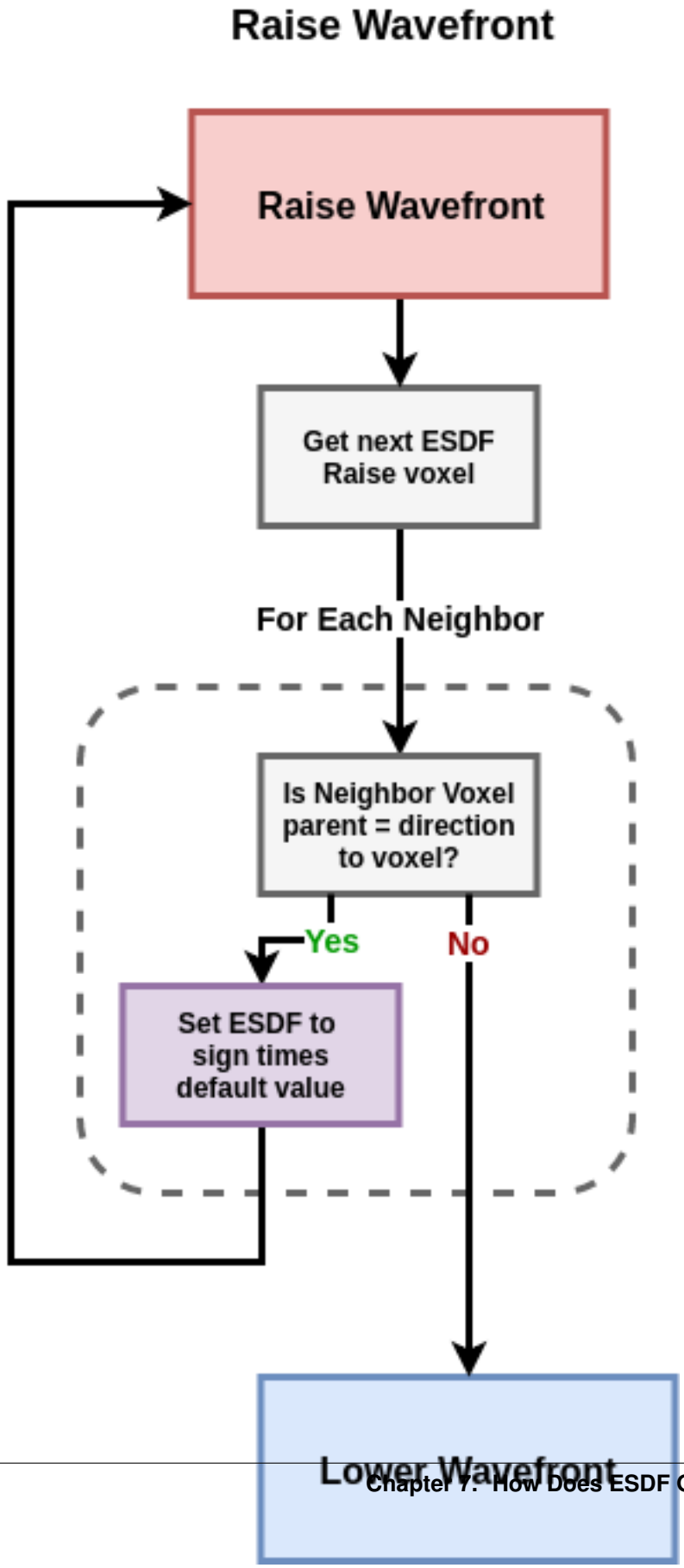
```
@inproceedings{oleynikova2017voxblox,
  author={Oleynikova, Helen and Taylor, Zachary and Fehr, Marius and Siegwart, Roland␣
↪and  Nieto, Juan},
  booktitle={IEEE/RSJ International Conference on Intelligent Robots and Systems␣
↪(IROS)},
  title={Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV␣
↪Planning},
  year={2017}
}
```

We have some system flowcharts below to make it easier to understand the general flow of data.

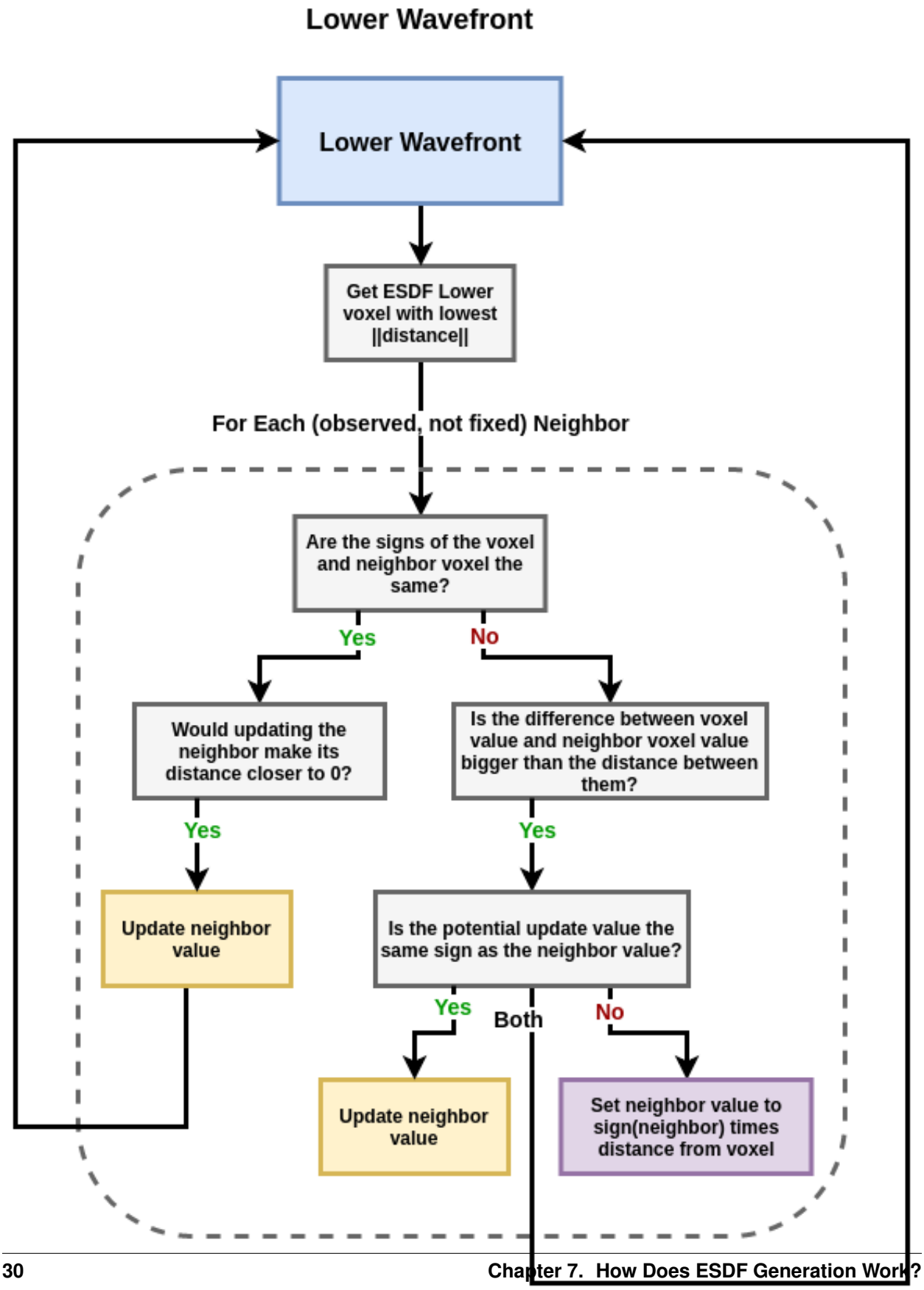
7.1.1 How TSDF values are propagated to ESDF



7.1.2 How the Raise Wavefront works



7.1.3 How the Lower Wavefront works



CHAPTER 8

Transformations in Voxblox

Voxblox uses active transforms and Hamilton quaternions. For further details on the notation used throughout the code see the minkindr wiki

CHAPTER 9

Contributing to Voxblox

These steps are only necessary if you plan on contributing to voxblox.

9.1 Code style

We follow the style and best practices listed in the Google C++ Style Guide.

9.1.1 Setting up the linter

This sets up a linter which checks if the code conforms to our style guide during commits.

First, install the dependencies listed here.

```
cd ~/catkin_ws/src/
git clone git@github.com:ethz-asl/linter.git
cd linter
echo ". $(realpath setup_linter.sh)" >> ~/.bashrc  # Or the matching file for
                                                   # your shell.
bash

# Initialize linter in voxblox repo
cd ~/catkin_ws/src/voxblox
init_linter_git_hooks
```

For more information about the linter visit ethz/linter

9.2 Modifying Voxblox

Here's some hints on how to extend voxblox to fit your needs...

9.2.1 Serialization

Serialization is currently implemented for:

- TSDF layers
- ESDF layers
- Occupancy layers

The following serialization tools are implemented:

- Store a layer to file
- Load layer from file
- Store a subset of the blocks of a layer to file
- Load blocks from file and add to a layer

How to add your own voxel/layer type

- Add your own voxel type and implement the `getVoxelType()`, e.g. `fancy_voxel.h`:

```
namespace voxblox {

// Used for serialization only.
namespace voxel_types {
 const std::string kYOUR_FANCY_VOXEL = "fancy_voxel"
}  // namespace voxel_types

template <>
inline std::string getVoxelType<YOUR_FANCY_VOXEL>() {
 return voxel_types::kYOUR_FANCY_VOXEL;
}

}  // namespace voxblox
```

- Implement the block (de)serialization functions for your voxel type, e.g. `fancy_block_serialization.cc`

```
namespace voxblox {

template <>
void Block<YOUR_FANCY_VOXEL>::DeserializeVoxelData(const BlockProto& proto,
                                                   YOUR_FANCY_VOXEL* voxels) {
// Your serialization code.
}

template <>
void Block<YOUR_FANCY_VOXEL>::SerializeVoxelData(const YOUR_FANCY_VOXEL* voxels,
                                                 BlockProto* proto) const {
// Your serialization code.
}

}  // namespace voxblox
```

- Create your own fancy_integrator.h, fancy_mesh_integrator.h, ...

  Have a look at the example package:

TODO(mfehr, helenol): add example package with a new voxel type

CHAPTER 10

Library API

10.1 Class Hierarchy

10.2 File Hierarchy

10.3 Full API

10.3.1 Namespaces

Namespace voxblox

These classes allocate a fixed size array and index it with a hash that is masked so that only its first N bits are non zero.

Contents

- *Detailed Description*
- *Namespaces*
- *Classes*
- *Enums*
- *Functions*
- *Typedefs*
- *Variables*

Detailed Description

This file contains a set of functions to visualize layers as pointclouds (or marker arrays) based on a passed-in function. This can be thought of as a fast rough approximation of a hash table. There are several advantages and some very significant disadvantages Advantages-

- Simple and blazing fast lockless thread-safe approximate sets
- Can be used to provide more fine grain locking of blocks for threading then simply locking the entire layer Disadvantages-
- Highly inefficient use of memory (allocates 2^N elements)
- Cannot discern between two different elements with the same hash
- **If the hash of two elements have the same first N elements of their hash, only one can be stored.** It also offers some specializations of functions as samples.

Namespaces

- *Namespace voxblox::io*
- *Namespace voxblox::test*
- *Namespace voxblox::timing*
- *Namespace voxblox::utils*
- *Namespace voxblox::voxel_types*

Classes

- *Struct AnyIndexHash*
- *Template Struct AnyIndexHashMapType*
- *Struct Color*
- *Struct EsdfIntegrator::Config*
- *Struct EsdfMap::Config*
- *Struct EsdfOccIntegrator::Config*
- *Struct EsdfVoxel*
- *Struct ICP::Config*
- *Struct IntensityVoxel*
- *Struct LongIndexHash*
- *Template Struct LongIndexHashMapType*
- *Struct Mesh*
- *Struct MeshIntegratorConfig*
- *Struct OccupancyIntegrator::Config*
- *Struct OccupancyMap::Config*
- *Struct OccupancyVoxel*

- *Struct TsdfIntegratorBase::Config*
- *Struct TsdfMap::Config*
- *Struct TsdfVoxel*
- *Template Class ApproxHashArray*
- *Template Class ApproxHashSet*
- *Template Class Block*
- *Template Class BucketQueue*
- *Class CameraModel*
- *Class ColorMap*
- *Class Cube*
- *Class Cylinder*
- *Class EsdfIntegrator*
- *Class EsdfMap*
- *Class EsdfOccIntegrator*
- *Class EsdfServer*
- *Class FastTsdfIntegrator*
- *Class GrayscaleColorMap*
- *Class ICP*
- *Class IntensityIntegrator*
- *Class IntensityServer*
- *Class InteractiveSlider*
- *Template Class Interpolator*
- *Class InverseGrayscaleColorMap*
- *Class InverseRainbowColorMap*
- *Class IronbowColorMap*
- *Template Class Layer*
- *Class MarchingCubes*
- *Class MergedTsdfIntegrator*
- *Template Class MeshIntegrator*
- *Class MeshLayer*
- *Template Class Neighborhood*
- *Class NeighborhoodLookupTables*
- *Class Object*
- *Class OccupancyIntegrator*
- *Class OccupancyMap*
- *Class Plane*

- *Class PlaneObject*
- *Class RainbowColorMap*
- *Class RayCaster*
- *Class SimpleTsdfIntegrator*
- *Class SimulationServer*
- *Class SimulationWorld*
- *Class Sphere*
- *Class ThreadSafeIndex*
- *Class Transformer*
- *Class TsdfIntegratorBase*
- *Class TsdfIntegratorFactory*
- *Class TsdfMap*
- *Class TsdfServer*

Enums

- *Enum ColorMode*
- *Enum Connectivity*
- *Enum MapDerializationAction*
- *Enum TsdfIntegratorType*

Functions

- *Template Function voxblox::aligned_shared*
- *Function voxblox::castRay*
- *Function voxblox::colorMsgToVoxblox*
- *Function voxblox::colorVoxbloxToMsg*
- *Function voxblox::convertMeshLayerToMesh*
- *Template Function voxblox::createColorPointcloudFromLayer(const Layer<VoxelType>&, const ShouldVisualizeVoxelColorFunctionType<VoxelType>&, pcl::PointCloud<pcl::PointXYZRGB> *)*
- *Template Function voxblox::createColorPointcloudFromLayer(const Layer<VoxelType>&, const ShouldVisualizeVoxelIntensityFunctionType<VoxelType>&, pcl::PointCloud<pcl::PointXYZI> *)*
- *Function voxblox::createConnectedMesh(const Mesh&, Mesh *, const FloatingPoint)*
- *Function voxblox::createConnectedMesh(const AlignedVector<Mesh::ConstPtr>&, Mesh *, const FloatingPoint)*
- *Function voxblox::createDistancePointcloudFromEsdfLayer*
- *Function voxblox::createDistancePointcloudFromEsdfLayerSlice*
- *Function voxblox::createDistancePointcloudFromTsdfLayer*

- *Function voxblox::createDistancePointcloudFromTsdfLayerSlice*
- *Function voxblox::createFreePointcloudFromEsdfLayer*
- *Function voxblox::createIntensityPointcloudFromIntensityLayer*
- *Template Function voxblox::createOccupancyBlocksFromLayer*
- *Function voxblox::createOccupancyBlocksFromOccupancyLayer*
- *Function voxblox::createOccupancyBlocksFromTsdfLayer*
- *Function voxblox::createPointcloudFromTsdfLayer*
- *Function voxblox::createSurfaceDistancePointcloudFromTsdfLayer*
- *Function voxblox::createSurfacePointcloudFromTsdfLayer*
- *Template Function voxblox::deserializeMsgToLayer(const voxblox_msgs::Layer&, Layer<VoxelType> *)*
- *Template Function voxblox::deserializeMsgToLayer(const voxblox_msgs::Layer&, const MapDerializationAction&, Layer<VoxelType> *)*
- *Template Function voxblox::evaluateLayerRmseAtPoses(const utils::VoxelEvaluationMode&, const Layer<VoxelType>&, const Layer<VoxelType>&, const std::vector<Transformation>&, std::vector<utils::VoxelEvaluationDetails> *, std::vector<std::pair<typename voxblox::Layer<VoxelType>::Ptr, typename voxblox::Layer<VoxelType>::Ptr>> *)*
- *Template Function voxblox::evaluateLayerRmseAtPoses(const utils::VoxelEvaluationMode&, const Layer<VoxelType>&, const Layer<VoxelType>&, const std::vector<Eigen::Matrix<float, 4, 4>, Eigen::aligned_allocator<Eigen::Matrix<float, 4, 4>>>&, std::vector<utils::VoxelEvaluationDetails> *, std::vector<std::pair<typename voxblox::Layer<VoxelType>::Ptr, typename voxblox::Layer<VoxelType>::Ptr>> *)*
- *Function voxblox::fillMarkerWithMesh*
- *Function voxblox::fillPointcloudWithMesh*
- *Function voxblox::generateVoxbloxMeshMsg*
- *Function voxblox::getBlockAndVoxelIndexFromGlobalVoxelIndex*
- *Function voxblox::getBlockIndexFromGlobalVoxelIndex*
- *Template Function voxblox::getCenterPointFromGridIndex*
- *Function voxblox::getEsdfIntegratorConfigFromRosParam*
- *Function voxblox::getEsdfMapConfigFromRosParam*
- *Function voxblox::getGlobalVoxelIndexFromBlockAndVoxelIndex*
- *Template Function voxblox::getGridIndexFromOriginPoint*
- *Template Function voxblox::getGridIndexFromPoint(const Point&)*
- *Template Function voxblox::getGridIndexFromPoint(const Point&, const FloatingPoint)*
- *Function voxblox::getHierarchicalIndexAlongRay*
- *Function voxblox::getICPConfigFromRosParam*
- *Function voxblox::getLocalFromGlobalVoxelIndex*
- *Template Function voxblox::getOriginPointFromGridIndex*
- *Template Function voxblox::getSurfaceDistanceAlongRay*
- *Function voxblox::getTsdfIntegratorConfigFromRosParam*

- *Function voxblox::getTsdfMapConfigFromRosParam*
- *Function voxblox::getVertexColor*
- *Template Function voxblox::getVoxelType*
- *Function voxblox::getVoxelType< EsdfVoxel >*
- *Function voxblox::getVoxelType< IntensityVoxel >*
- *Function voxblox::getVoxelType< OccupancyVoxel >*
- *Function voxblox::getVoxelType< TsdfVoxel >*
- *Function voxblox::grayColorMap*
- *Function voxblox::heightColorFromVertex*
- *Function voxblox::isPowerOfTwo*
- *Function voxblox::lambertColorFromColorAndNormal*
- *Function voxblox::lambertColorFromNormal*
- *Function voxblox::lambertShading*
- *Function voxblox::logOddsFromProbability*
- *Template Function voxblox::mergeLayerAintoLayerB(const Layer<VoxelType>&, const Transformation&, Layer<VoxelType> *, bool)*
- *Template Function voxblox::mergeLayerAintoLayerB(const Layer<VoxelType>&, Layer<VoxelType> *)*
- *Function voxblox::mergeVoxelAIntoVoxelB(const TsdfVoxel&, TsdfVoxel *)*
- *Template Function voxblox::mergeVoxelAIntoVoxelB(const VoxelType&, VoxelType *)*
- *Function voxblox::mergeVoxelAIntoVoxelB(const EsdfVoxel&, EsdfVoxel *)*
- *Function voxblox::mergeVoxelAIntoVoxelB(const OccupancyVoxel&, OccupancyVoxel *)*
- *Template Function voxblox::naiveTransformLayer*
- *Function voxblox::normalColorFromNormal*
- *Function voxblox::outputMeshAsPly*
- *Function voxblox::outputMeshLayerAsPly(const std::string&, const MeshLayer&)*
- *Function voxblox::outputMeshLayerAsPly(const std::string&, const bool, const MeshLayer&)*
- *Function voxblox::pointcloudToPclXYZ*
- *Function voxblox::pointcloudToPclXYZI*
- *Function voxblox::pointcloudToPclXYZRGB*
- *Function voxblox::probabilityFromLogOdds*
- *Function voxblox::rainbowColorMap*
- *Function voxblox::randomColor*
- *Function voxblox::recolorVoxbloxMeshMsgByIntensity*
- *Template Function voxblox::resampleLayer*
- *Template Function voxblox::serializeLayerAsMsg*
- *Function voxblox::signum*

- *Function voxblox::toConnectedPCLPolygonMesh*
- *Function voxblox::toPCLPolygonMesh*
- *Function voxblox::toSimplifiedPCLPolygonMesh*
- *Template Function voxblox::transformLayer*
- *Function voxblox::transformPointcloud*
- *Function voxblox::visualizeDistanceIntensityEsdfVoxels*
- *Function voxblox::visualizeDistanceIntensityEsdfVoxelsSlice*
- *Function voxblox::visualizeDistanceIntensityTsdfVoxels*
- *Function voxblox::visualizeDistanceIntensityTsdfVoxelsNearSurface*
- *Function voxblox::visualizeDistanceIntensityTsdfVoxelsSlice*
- *Function voxblox::visualizeFreeEsdfVoxels*
- *Function voxblox::visualizeIntensityVoxels*
- *Function voxblox::visualizeNearSurfaceTsdfVoxels*
- *Function voxblox::visualizeOccupiedOccupancyVoxels*
- *Function voxblox::visualizeOccupiedTsdfVoxels*
- *Function voxblox::visualizeTsdfVoxels*

Typedefs

- *Typedef voxblox::AlignedDeque*
- *Typedef voxblox::AlignedLayerAndErrorLayer*
- *Typedef voxblox::AlignedLayerAndErrorLayers*
- *Typedef voxblox::AlignedList*
- *Typedef voxblox::AlignedQueue*
- *Typedef voxblox::AlignedStack*
- *Typedef voxblox::AlignedVector*
- *Typedef voxblox::AnyIndex*
- *Typedef voxblox::BlockIndex*
- *Typedef voxblox::BlockIndexList*
- *Typedef voxblox::Colors*
- *Typedef voxblox::FloatingPoint*
- *Typedef voxblox::GlobalIndex*
- *Typedef voxblox::GlobalIndexVector*
- *Typedef voxblox::HierarchicalIndex*
- *Typedef voxblox::HierarchicalIndexMap*
- *Typedef voxblox::HierarchicalIndexSet*
- *Typedef voxblox::IndexElement*

- *Typedef voxblox::IndexSet*
- *Typedef voxblox::IndexVector*
- *Typedef voxblox::InterpIndexes*
- *Typedef voxblox::InterpTable*
- *Typedef voxblox::InterpVector*
- *Typedef voxblox::Label*
- *Typedef voxblox::LabelConfidence*
- *Typedef voxblox::Labels*
- *Typedef voxblox::LongIndex*
- *Typedef voxblox::LongIndexElement*
- *Typedef voxblox::LongIndexSet*
- *Typedef voxblox::LongIndexVector*
- *Typedef voxblox::Point*
- *Typedef voxblox::Pointcloud*
- *Typedef voxblox::PointsMatrix*
- *Typedef voxblox::Quaternion*
- *Typedef voxblox::Ray*
- *Typedef voxblox::Rotation*
- *Typedef voxblox::ShouldVisualizeVoxelColorFunctionType*
- *Typedef voxblox::ShouldVisualizeVoxelFunctionType*
- *Typedef voxblox::ShouldVisualizeVoxelIntensityFunctionType*
- *Typedef voxblox::SignedIndex*
- *Typedef voxblox::SquareMatrix*
- *Typedef voxblox::Transformation*
- *Typedef voxblox::Triangle*
- *Typedef voxblox::TriangleVector*
- *Typedef voxblox::VertexIndex*
- *Typedef voxblox::VertexIndexList*
- *Typedef voxblox::VoxelIndex*
- *Typedef voxblox::VoxelIndexList*
- *Typedef voxblox::VoxelKey*

Variables

- *Variable voxblox::kDefaultMaxIntensity*
- *Variable voxblox::kEpsilon*
- *Variable voxblox::kFloatEpsilon*

- *Variable voxblox::kNumTsdfIntegratorTypes*
- *Variable voxblox::kTsdfIntegratorTypeNames*
- *Variable voxblox::kUnitCubeDiagonalLength*

Namespace voxblox::io

Contents

- *Classes*
- *Enums*
- *Functions*

Classes

- *Class PlyWriter*

Enums

- *Enum PlyOutputTypes*

Functions

- *Template Function voxblox::io::convertLayerToMesh(const Layer<VoxelType>&, voxblox::Mesh *, const bool, const FloatingPoint)*
- *Template Function voxblox::io::convertLayerToMesh(const Layer<VoxelType>&, const MeshIntegratorConfig&, voxblox::Mesh *, const bool, const FloatingPoint)*
- *Template Function voxblox::io::convertVoxelGridToPointCloud(const Layer<VoxelType>&, const float, const float, voxblox::Mesh *)*
- *Template Function voxblox::io::convertVoxelGridToPointCloud(const Layer<VoxelType>&, const float, voxblox::Mesh *)*
- *Function voxblox::io::getColorFromVoxel(const TsdfVoxel&, const float, const float, Color *)*
- *Template Function voxblox::io::getColorFromVoxel(const VoxelType&, const float, const float, Color *)*
- *Function voxblox::io::getColorFromVoxel(const EsdfVoxel&, const float, const float, Color *)*
- *Template Function voxblox::io::LoadBlocksFromFile(const std::string&, typename Layer<VoxelType>::BlockMergingStrategy, Layer<VoxelType> *)*
- *Template Function voxblox::io::LoadBlocksFromFile(const std::string&, typename Layer<VoxelType>::BlockMergingStrategy, bool, Layer<VoxelType> *)*
- *Template Function voxblox::io::LoadLayer(const std::string&, const bool, typename Layer<VoxelType>::Ptr *)*
- *Template Function voxblox::io::LoadLayer(const std::string&, typename Layer<VoxelType>::Ptr *)*
- *Template Function voxblox::io::outputLayerAsPly*

- *Template Function voxblox::io::SaveLayer*
- *Template Function voxblox::io::SaveLayerSubset*

Namespace voxblox::test

Contents

- *Classes*
- *Functions*

Classes

- *Template Class LayerTest*

Functions

- *Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, OccupancyVoxel *)*
- *Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, TsdfVoxel *)*
- *Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, EsdfVoxel *)*
- *Template Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, VoxelType *)*
- *Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, IntensityVoxel *)*
- *Template Function voxblox::test::SetUpTestLayer(const IndexElement, Layer<VoxelType> *)*
- *Template Function voxblox::test::SetUpTestLayer(const IndexElement, const IndexElement, Layer<VoxelType> *)*

Namespace voxblox::timing

Contents

- *Classes*
- *Typedefs*

Classes

- *Struct TimerMapValue*
- *Template Class Accumulator*
- *Class DummyTimer*
- *Class Timer*
- *Class Timing*

Typedefs

- *Typedef voxblox::timing::DebugTimer*

Namespace voxblox::utils

Contents

- *Classes*
- *Enums*
- *Functions*

Classes

- *Struct VoxelEvaluationDetails*

Enums

- *Enum VoxelEvaluationMode*
- *Enum VoxelEvaluationResult*

Functions

- *Template Function voxblox::utils::centerBlocksOfLayer*
- *Template Function voxblox::utils::clearSphereAroundPoint*
- *Template Function voxblox::utils::computeMapBoundsFromLayer*
- *Template Function voxblox::utils::computeVoxelError*
- *Template Function voxblox::utils::evaluateLayersRmse(const Layer<VoxelType>&, const Layer<VoxelType>&, const VoxelEvaluationMode&, VoxelEvaluationDetails *, Layer<VoxelType> *)*
- *Template Function voxblox::utils::evaluateLayersRmse(const Layer<VoxelType>&, const Layer<VoxelType>&)*
- *Template Function voxblox::utils::fillSphereAroundPoint*
- *Template Function voxblox::utils::getAndAllocateSphereAroundPoint*
- *Function voxblox::utils::getColorIfValid(const TsdfVoxel&, const FloatingPoint, Color *)*
- *Function voxblox::utils::getColorIfValid(const EsdfVoxel&, const FloatingPoint, Color *)*
- *Template Function voxblox::utils::getColorIfValid(const VoxelType&, const FloatingPoint, Color *)*
- *Template Function voxblox::utils::getSdfIfValid(const VoxelType&, const FloatingPoint, FloatingPoint *)*
- *Function voxblox::utils::getSdfIfValid(const TsdfVoxel&, const FloatingPoint, FloatingPoint *)*
- *Function voxblox::utils::getSdfIfValid(const EsdfVoxel&, const FloatingPoint, FloatingPoint *)*
- *Template Function voxblox::utils::getSphereAroundPoint*

- *Function voxblox::utils::getVoxelSdf(const TsdfVoxel&)*
- *Function voxblox::utils::getVoxelSdf(const EsdfVoxel&)*
- *Template Function voxblox::utils::getVoxelSdf(const VoxelType&)*
- *Function voxblox::utils::isObservedVoxel(const EsdfVoxel&)*
- *Template Function voxblox::utils::isObservedVoxel(const VoxelType&)*
- *Function voxblox::utils::isObservedVoxel(const TsdfVoxel&)*
- *Template Function voxblox::utils::isSameBlock*
- *Template Function voxblox::utils::isSameLayer*
- *Template Function voxblox::utils::isSameVoxel(const VoxelType&, const VoxelType&)*
- *Function voxblox::utils::isSameVoxel(const TsdfVoxel&, const TsdfVoxel&)*
- *Function voxblox::utils::isSameVoxel(const EsdfVoxel&, const EsdfVoxel&)*
- *Function voxblox::utils::isSameVoxel(const OccupancyVoxel&, const OccupancyVoxel&)*
- *Function voxblox::utils::readProtoMsgCountToStream*
- *Function voxblox::utils::readProtoMsgFromStream*
- *Function voxblox::utils::setVoxelSdf(const FloatingPoint, TsdfVoxel *)*
- *Function voxblox::utils::setVoxelSdf(const FloatingPoint, EsdfVoxel *)*
- *Template Function voxblox::utils::setVoxelSdf(const FloatingPoint, VoxelType *)*
- *Function voxblox::utils::setVoxelWeight(const FloatingPoint, TsdfVoxel *)*
- *Function voxblox::utils::setVoxelWeight(const FloatingPoint, EsdfVoxel *)*
- *Template Function voxblox::utils::setVoxelWeight(const FloatingPoint, VoxelType *)*
- *Function voxblox::utils::writeProtoMsgCountToStream*
- *Function voxblox::utils::writeProtoMsgToStream*

Namespace voxblox::voxel_types

Used for serialization only.

Contents

- *Variables*

Variables

- *Variable voxblox::voxel_types::kEsdf*
- *Variable voxblox::voxel_types::kIntensity*
- *Variable voxblox::voxel_types::kNotSerializable*
- *Variable voxblox::voxel_types::kOccupancy*
- *Variable voxblox::voxel_types::kTsdf*

Namespace voxblox_rviz_plugin

Contents

- *Classes*

Classes

- *Class VoxbloxMeshDisplay*
- *Class VoxbloxMeshVisual*

10.3.2 Classes and Structs

Struct AnyIndexHash

- Defined in *File block_hash.h*

Struct Documentation

struct AnyIndexHash

Performs deco hashing on block indexes.

Based on recommendations of "Investigating the impact of Suboptimal Hashing Functions" by L. Buckley et al.

Public Functions

std::size_t **operator()**(**const** *AnyIndex* &*index*) **const**

Public Static Attributes

constexpr size_t **sl** = 17191

number was arbitrarily chosen with no good justification

constexpr size_t **sl2** = *sl* * *sl*

Template Struct AnyIndexHashMapType

- Defined in *File block_hash.h*

Struct Documentation

template <typename *ValueType*>
struct AnyIndexHashMapType

Public Types

typedef std::unordered_map<*AnyIndex*, ValueType, *AnyIndexHash*, std::equal_to<*AnyIndex*>, Eigen::aligned_allocator<std::p

Struct Color

- Defined in *File common.h*

Struct Documentation

struct Color

Public Functions

Color()

Color(uint8_t *_r*, uint8_t *_g*, uint8_t *_b*)

Color(uint8_t *_r*, uint8_t *_g*, uint8_t *_b*, uint8_t *_a*)

Public Members

uint8_t **r**

uint8_t **g**

uint8_t **b**

uint8_t **a**

Public Static Functions

static *Color* **blendTwoColors**(**const** *Color* &*first_color*, *FloatingPoint* *first_weight*, **const** *Color* &*second_color*, *FloatingPoint* *second_weight*)

static const *Color* **White**()

static const *Color* **Black**()

static const *Color* **Gray**()

static const *Color* **Red**()

static const *Color* **Green**()

static const *Color* **Blue**()

static const *Color* **Yellow**()

static const *Color* **Orange**()

static const *Color* **Purple**()

static const *Color* **Teal**()

static const *Color* **Pink**()

Struct EsdfIntegrator::Config

- Defined in *File esdf_integrator.h*

Nested Relationships

This struct is a nested type of *Class EsdfIntegrator*.

Struct Documentation

struct Config

Public Members

bool **full_euclidean_distance** = false
Whether to use full euclidean distance (true) or quasi-euclidean (false).

Full euclidean is slightly more accurate (up to 8% in the worst case) but slower.

FloatingPoint **max_distance_m** = 2.0
Maximum distance to calculate the actual distance to.

Any values above this will be set to default_distance_m.

FloatingPoint **min_distance_m** = 0.2
Should mirror (or be smaller than) truncation distance in tsdf integrator.

FloatingPoint **default_distance_m** = 2.0
Default distance set for unknown values and values > max_distance_m.

FloatingPoint **min_diff_m** = 0.001
For cheaper but less accurate map updates: the minimum difference in a voxel distance, before the change is propagated.

float **min_weight** = 1e-6
Minimum weight to consider a TSDF value seen at.

int **num_buckets** = 20
Number of buckets for the bucketed priority queue.

bool **multi_queue** = false
Whether to push stuff to the open queue multiple times, with updated distances.

bool **add_occupied_crust** = false
Whether to add an outside layer of occupied voxels.

Basically just sets all unknown voxels in the allocated blocks to occupied. Try to only use this for batch processing, otherwise look into addNewRobotPosition below, which uses clear spheres.

FloatingPoint **clear_sphere_radius** = 1.5
For marking unknown space around a robot as free or occupied, these are the radiuses used around each robot position.

FloatingPoint **occupied_sphere_radius** = 5.0

Struct EsdfMap::Config

- Defined in *File esdf_map.h*

Nested Relationships

This struct is a nested type of *Class EsdfMap*.

Struct Documentation

struct Config

Public Members

FloatingPoint **esdf_voxel_size** = 0.2

size_t **esdf_voxels_per_side** = 16u

Struct EsdfOccIntegrator::Config

- Defined in *File esdf_occ_integrator.h*

Nested Relationships

This struct is a nested type of *Class EsdfOccIntegrator*.

Struct Documentation

struct Config

Public Members

FloatingPoint **max_distance_m** = 2.0
Maximum distance to calculate the actual distance to.

Any values above this will be set to default_distance_m.

FloatingPoint **default_distance_m** = 2.0
Default distance set for unknown values and values > max_distance_m.

int **num_buckets** = 20
Number of buckets for the bucketed priority queue.

Struct EsdfVoxel

- Defined in *File voxel.h*

Struct Documentation

struct EsdfVoxel

Public Members

float **distance** = 0.0f

bool **observed** = false

bool **hallucinated** = false
Whether the voxel was copied from the TSDF (false) or created from a pose or some other source (true).

This member is not serialized!!!

bool **in_queue** = false

bool **fixed** = false

Eigen::Vector3i **parent** = Eigen::Vector3i::Zero()
Relative direction toward parent.

If itself, then either uninitialized or in the fixed frontier.

Struct ICP::Config

- Defined in *File icp.h*

Nested Relationships

This struct is a nested type of *Class ICP*.

Struct Documentation

struct Config
Contains all the information needed to setup the *ICP* class.

Public Members

bool **refine_roll_pitch** = false

int **mini_batch_size** = 20
Number of points used in each alignment step.

To allow simple threading the *ICP* process is split up into a large number of separate alignments performed on small pointclouds. This parameter dictates how many points are used in each "mini batch". The result are then combined weighting them by an estimate of the information gained by the alignment.

FloatingPoint **min_match_ratio** = 0.8
Ratio of points that must lie within the truncation distance of an allocated voxel.

FloatingPoint **subsample_keep_ratio** = 0.5
Ratio of points used in the *ICP* matching.

FloatingPoint **inital_translation_weighting** = 100.0
Weighting applied to the translational component of the initial guess.

Very roughly corresponds to the inverse covariance of the initial guess multiplied by the variance in a measured points accuracy.

FloatingPoint **inital_rotation_weighting** = 100.0
Weighting applied to the rotational component of the initial guess.

See inital_translation_weighting for further details

size_t **num_threads** = std::thread::hardware_concurrency()

Struct IntensityVoxel

- Defined in *File voxel.h*

Struct Documentation

struct IntensityVoxel

Public Members

float **intensity** = 0.0f

float **weight** = 0.0f

Struct LongIndexHash

- Defined in *File block_hash.h*

Struct Documentation

struct LongIndexHash
Hash for large index values, see *AnyIndexHash*.

Public Functions

std::size_t **operator()**(**const** *LongIndex* &*index*) **const**

Public Static Attributes

constexpr size_t **sl** = 17191

constexpr size_t **sl2** = *sl* * *sl*

Template Struct LongIndexHashMapType

- Defined in *File block_hash.h*

Struct Documentation

template <typename *ValueType*>
struct LongIndexHashMapType

Public Types

typedef std::unordered_map<*LongIndex*, ValueType, *LongIndexHash*, std::equal_to<*LongIndex*>, Eigen::aligned_allocator<st

Struct Mesh

- Defined in *File mesh.h*

Struct Documentation

struct Mesh
Holds the vertex, normals, color and triangle index information of a mesh.

Public Types

typedef std::shared_ptr<*Mesh*> **Ptr**

typedef std::shared_ptr<**const** *Mesh*> **ConstPtr**

Public Functions

Mesh()

Mesh(*FloatingPoint* *_block_size*, **const** *Point* &*_origin*)

virtual ~Mesh()

bool **hasVertices**() **const**

bool **hasNormals**() **const**

bool **hasColors**() **const**

bool **hasTriangles**() **const**

size_t **size**() **const**

void **clear**()

void **clearTriangles**()

void **clearNormals**()

void **clearColors**()

void **resize**(**const** size_t *size*, **const** bool *has_normals* = true, **const** bool *has_colors* = true, **const** bool *has_indices* = true)

void **reserve**(**const** size_t *size*, **const** bool *has_normals* = true, **const** bool *has_colors* = true, **const** bool *has_triangles* = true)

void **colorizeMesh**(**const** *Color* &*new_color*)

void **concatenateMesh**(**const** *Mesh* &*other_mesh*)

Public Members

Pointcloud **vertices**

VertexIndexList **indices**

Pointcloud **normals**

Colors **colors**

FloatingPoint **block_size**

Point **origin**

bool **updated**

Public Static Attributes

constexpr *FloatingPoint* **kInvalidBlockSize** = -1.0

Struct MeshIntegratorConfig

- Defined in *File mesh_integrator.h*

Struct Documentation

struct MeshIntegratorConfig

Public Members

bool **use_color** = true

float **min_weight** = 1e-4

size_t **integrator_threads** = std::thread::hardware_concurrency()

Struct OccupancyIntegrator::Config

- Defined in *File occupancy_integrator.h*

Nested Relationships

This struct is a nested type of *Class OccupancyIntegrator*.

Struct Documentation

struct Config

Public Members

float **probability_hit** = 0.65f

float **probability_miss** = 0.4f

float **threshold_min** = 0.12f

float **threshold_max** = 0.97f

float **threshold_occupancy** = 0.7f

FloatingPoint **min_ray_length_m** = 0.1

FloatingPoint **max_ray_length_m** = 5.0

Struct OccupancyMap::Config

- Defined in *File occupancy_map.h*

Nested Relationships

This struct is a nested type of *Class OccupancyMap*.

Struct Documentation

struct Config

Public Members

FloatingPoint **occupancy_voxel_size** = 0.2

size_t **occupancy_voxels_per_side** = 16u

Struct OccupancyVoxel

- Defined in *File voxel.h*

Struct Documentation

struct OccupancyVoxel

Public Members

float **probability_log** = 0.0f

bool **observed** = false

Struct TimerMapValue

- Defined in *File timing.h*

Struct Documentation

struct TimerMapValue

Public Functions

TimerMapValue()

Public Members

Accumulator<double, double, 50> **acc_**
Create an accumulator with specified window size.

Struct TsdfIntegratorBase::Config

- Defined in *File tsdf_integrator.h*

Nested Relationships

This struct is a nested type of *Class TsdfIntegratorBase*.

Struct Documentation

struct Config

Public Members

float **default_truncation_distance** = 0.1

float **max_weight** = 10000.0

bool **voxel_carving_enabled** = true

FloatingPoint **min_ray_length_m** = 0.1

FloatingPoint **max_ray_length_m** = 5.0

bool **use_const_weight** = false

bool **allow_clear** = true

bool **use_weight_dropoff** = true

bool **use_sparsity_compensation_factor** = false

float **sparsity_compensation_factor** = 1.0f

size_t **integrator_threads** = std::thread::hardware_concurrency()

bool **enable_anti_grazing** = false
merge integrator specific

float **start_voxel_subsampling_factor** = 2.0f
fast integrator specific

int **max_consecutive_ray_collisions** = 2
fast integrator specific

int **clear_checks_every_n_frames** = 1
fast integrator specific

float **max_integration_time_s** = std::numeric_limits<float>::max()
fast integrator specific

Struct TsdfMap::Config

- Defined in *File tsdf_map.h*

Nested Relationships

This struct is a nested type of *Class TsdfMap*.

Struct Documentation

struct Config

Public Members

FloatingPoint **tsdf_voxel_size** = 0.2

size_t **tsdf_voxels_per_side** = 16u

Struct TsdfVoxel

- Defined in *File voxel.h*

Struct Documentation

struct TsdfVoxel

Public Members

float **distance** = 0.0f

float **weight** = 0.0f

Color **color**

Struct VoxelEvaluationDetails

- Defined in *File evaluation_utils.h*

Struct Documentation

struct VoxelEvaluationDetails

Public Functions

std::string **toString**() **const**

Public Members

FloatingPoint **rmse** = 0.0

FloatingPoint **max_error** = 0.0

FloatingPoint **min_error** = 0.0

size_t **num_evaluated_voxels** = 0u

size_t **num_ignored_voxels** = 0u

size_t **num_overlapping_voxels** = 0u

size_t **num_non_overlapping_voxels** = 0u

Template Class ApproxHashArray

- Defined in *File approx_hash_array.h*

Class Documentation

template <size_t *unmasked_bits*, typename *StoredElement*, typename *IndexType*, typename *IndexTypeHasher*>
class ApproxHashArray

Basic container, give in an index and get the element that was stored there.

There are 2^unmasked_bits elements in the container, which element is returned depends on your hashing function. Uses at least 2^unmasked_bits * sizeof(StoreElement) bytes of ram

Public Functions

StoredElement &**get**(**const** size_t &*hash*)

StoredElement &**get**(**const** IndexType &*index*, size_t *&*hash*)

StoredElement &**get**(**const** IndexType &*index*)

Template Class ApproxHashSet

- Defined in *File approx_hash_array.h*

Class Documentation

template <size_t *unmasked_bits*, size_t *full_reset_threshold*, typename *IndexType*, typename *IndexTypeHasher*>
class ApproxHashSet

Acts as a fast and thread safe set, with the serious limitation of both false-negatives and false positives being possible.

A false positive occurs if two different elements have the same hash, and the other element was already added to the set. A false negative occurs if an element was removed to add another element with the same masked hash. The chance of this happening is inversely proportional to 2^unmasked_bits. Uses at least (2^unmasked_bits + full_reset_threshold) * sizeof(StoreElement) bytes of ram. Note that the reset function is not thread safe.

Public Functions

ApproxHashSet()

bool **isHashCurrentlyPresent**(**const** size_t &*hash*)

Returns true if an element with the same hash is currently in the set, false otherwise.

Note due to the masking of bits, many elements that were previously inserted into the *ApproxHashSet* have been overwritten by other values.

bool **isHashCurrentlyPresent**(**const** IndexType &*index*, size_t **hash*)

bool **isHashCurrentlyPresent**(**const** IndexType &*index*)

bool **replaceHash**(**const** size_t &*hash*)

Returns true if it replaced the element in the masked_hash's with the hash of the given element.

Returns false if this hash was already there and no replacement was needed. THIS IS THE MOST EXPENSIVE FUNCTION IN ALL OF VOXBLOX. PROILE AND TEST AFTER EVEN THE MOST SUPERFICIAL CHANGE !!! Also note that while the below layout of ifs and variables may not appear the most efficient the compiler seems to do some black magic with it that makes it come out ahead of other formulations.

bool **replaceHash**(**const** IndexType &*index*, size_t **hash*)

bool **replaceHash**(**const** IndexType &*index*)

void **resetApproxSet**()

If unmasked_bits is large, the array takes a lot of memory, this makes clearing it slow.

However offsetting which bin hashes are placed into has the same effect. Once we run out of room to offset by (determined by full_reset_threshold) we clear the memory). This function is not thread safe.

Template Class Block

- Defined in *File block.h*

Class Documentation

template <typename *VoxelType*>
class Block

An nxnxn container holding VoxelType.

It is aware of its 3D position and contains functions for accessing voxels by position and index

Public Types

typedef std::shared_ptr<*Block*<VoxelType>> **Ptr**

typedef std::shared_ptr<**const** *Block*<VoxelType>> **ConstPtr**

Public Functions

Block(size_t *voxels_per_side*, *FloatingPoint* *voxel_size*, **const** *Point* &*origin*)

Block(**const** BlockProto &*proto*)

~Block()

size_t **computeLinearIndexFromVoxelIndex**(**const** *VoxelIndex* &*index*) **const**

VoxelIndex **computeTruncatedVoxelIndexFromCoordinates**(**const** *Point* &*coords*) **const**

NOTE: This function is dangerous, it will truncate the voxel index to an index that is within this block if you pass a coordinate outside the range of this block.

Try not to use this function if there is an alternative to directly address the voxels via precise integer indexing math.

VoxelIndex **computeVoxelIndexFromCoordinates**(**const** *Point* &*coords*) **const**

NOTE: This function is also dangerous, use in combination with *Block::isValidVoxelIndex* function.

This function doesn't truncate the voxel index to the [0, voxels_per_side] range when the coordinate is outside the range of this block, unlike the function above.

size_t **computeLinearIndexFromCoordinates**(**const** *Point* &*coords*) **const**

NOTE: This function is dangerous, it will truncate the voxel index to an index that is within this block if you pass a coordinate outside the range of this block.

Try not to use this function if there is an alternative to directly address the voxels via precise integer indexing math.

Point **computeCoordinatesFromLinearIndex**(size_t *linear_index*) **const**

Returns the CENTER point of the voxel.

Point **computeCoordinatesFromVoxelIndex**(**const** *VoxelIndex* &*index*) **const**

Returns the CENTER point of the voxel.

VoxelIndex **computeVoxelIndexFromLinearIndex**(size_t *linear_index*) **const**

const VoxelType &**getVoxelByLinearIndex**(size_t *index*) **const**

Accessors to actual blocks.

const VoxelType &**getVoxelByVoxelIndex**(**const** *VoxelIndex* &*index*) **const**

const VoxelType &**getVoxelByCoordinates**(**const** *Point* &*coords*) **const**

NOTE: This functions is dangerous, it will truncate the voxel index to an index that is within this block if you pass a coordinate outside the range of this block.

Try not to use this function if there is an alternative to directly address the voxels via precise integer indexing math.

VoxelType &**getVoxelByCoordinates**(**const** *Point* &*coords*)

NOTE: This functions is dangerous, it will truncate the voxel index to an index that is within this block if you pass a coordinate outside the range of this block.

Try not to use this function if there is an alternative to directly address the voxels via precise integer indexing math.

VoxelType ***getVoxelPtrByCoordinates**(**const** *Point* &*coords*)

NOTE: This functions is dangerous, it will truncate the voxel index to an index that is within this block if you pass a coordinate outside the range of this block.

Try not to use this function if there is an alternative to directly address the voxels via precise integer indexing math.

const VoxelType ***getVoxelPtrByCoordinates**(**const** *Point* &*coords*) **const**

VoxelType &**getVoxelByLinearIndex**(size_t *index*)

VoxelType &**getVoxelByVoxelIndex**(**const** *VoxelIndex* &*index*)

bool **isValidVoxelIndex**(**const** *VoxelIndex* &*index*) **const**

bool **isValidLinearIndex**(size_t *index*) **const**

BlockIndex **block_index**() **const**

size_t **voxels_per_side**() **const**

FloatingPoint **voxel_size**() **const**

FloatingPoint **voxel_size_inv**() **const**

size_t **num_voxels**() **const**

Point **origin**() **const**

void **setOrigin**(**const** *Point* &*new_origin*)

FloatingPoint **block_size**() **const**

bool **has_data**() **const**

bool **updated**() **const**

std::atomic<bool> &**updated**()

bool &**has_data**()

void **set_updated**(bool *updated*)

void **set_has_data**(bool *has_data*)

void **getProto**(BlockProto **proto*) **const**

void **serializeToIntegers**(std::vector<uint32_t> *data) **const**

void **deserializeFromIntegers**(**const** std::vector<uint32_t> &data)

void **mergeBlock**(**const** *Block*<VoxelType> &other_block)

size_t **getMemorySize**() **const**

Protected Attributes

std::unique_ptr<VoxelType[]> **voxels_**

size_t **num_voxels_**

bool **has_data_**
Is set to true if any one of the voxels in this block received an update.

Template Class BucketQueue

- Defined in *File bucket_queue.h*

Class Documentation

template <typename *T*>
class BucketQueue
Bucketed priority queue, mostly following L.

Yatziv et al in O(N) Implementation of the Fast Marching Algorithm, though skipping the circular aspect (don't care about a bit more memory used for this).

Public Functions

BucketQueue()

BucketQueue(int *num_buckets*, double *max_val*)

void **setNumBuckets**(int *num_buckets*, double *max_val*)
WARNING: will CLEAR THE QUEUE!

void **push**(**const** T &*key*, double *value*)

void **pop**()

T **front**()

bool **empty**()

void **clear**()

Class CameraModel

- Defined in *File camera_model.h*

Class Documentation

class CameraModel

Virtual camera model for use in simulating a systems view.

Public Functions

CameraModel()

virtual ~CameraModel()

void **setIntrinsicsFromFocalLength**(**const** Eigen::Matrix<*FloatingPoint*, 2, 1> &*resolution*, double *focal_length*, double *min_distance*, double *max_distance*)

Set up the camera model, intrinsics and extrinsics.

void **setIntrinsicsFromFoV**(double *horizontal_fov*, double *vertical_fov*, double *min_distance*, double *max_distance*)

void **setExtrinsics**(**const** *Transformation* &*T_C_B*)

Transformation **getCameraPose**() **const**

Get and set the current poses for the camera (should be called after the camera is properly set up).

Transformation **getBodyPose**() **const**

void **setCameraPose**(**const** *Transformation* &*cam_pose*)

Set camera pose actually computes the new bounding plane positions.

void **setBodyPose**(**const** *Transformation* &*body_pose*)

void **getAabb**(*Point* *aabb_min, *Point* *aabb_max) **const**

Check whether a point belongs in the current view.

bool **isPointInView**(**const** *Point* &*point*) **const**

const AlignedVector<*Plane*> &**getBoundingPlanes**() **const**

Accessor functions for visualization (or other reasons).

Bounding planes are returned in the global coordinate frame.

void **getBoundingLines**(AlignedVector<*Point*> **lines*) **const**

Gives all the bounding lines of the planes in connecting order, expressed in the global coordinate frame.

void **getFarPlanePoints**(AlignedVector<*Point*> **points*) **const**

Get the 3 points definining the plane at the back (far end) of the camera frustum.

Expressed in global coordinates.

Class ColorMap

- Defined in *File color_maps.h*

Inheritance Relationships

Derived Types

- `public voxblox::GrayscaleColorMap` (*Class GrayscaleColorMap*)
- `public voxblox::InverseGrayscaleColorMap` (*Class InverseGrayscaleColorMap*)
- `public voxblox::InverseRainbowColorMap` (*Class InverseRainbowColorMap*)
- `public voxblox::IronbowColorMap` (*Class IronbowColorMap*)
- `public voxblox::RainbowColorMap` (*Class RainbowColorMap*)

Class Documentation

class ColorMap

Subclassed by *voxblox::GrayscaleColorMap*, *voxblox::InverseGrayscaleColorMap*, *voxblox::InverseRainbowColorMap*, *voxblox::IronbowColorMap*, *voxblox::RainbowColorMap*

Public Functions

ColorMap()

virtual ~ColorMap()

void **setMinValue**(float *min_value*)

void **setMaxValue**(float *max_value*)

virtual *Color* **colorLookup**(float *value*) **const** = 0

Protected Attributes

float **min_value_**

float **max_value_**

Class Cube

- Defined in *File objects.h*

Inheritance Relationships

Base Type

- `public voxblox::Object` (*Class Object*)

Class Documentation

class Cube : **public** voxblox::*Object*

Public Functions

Cube(**const** *Point* &*center*, **const** *Point* &*size*)

Cube(**const** *Point* &*center*, **const** *Point* &*size*, **const** *Color* &*color*)

virtual *FloatingPoint* **getDistanceToPoint**(**const** *Point* &*point*) **const**
Map-building accessors.

virtual bool **getRayIntersection**(**const** *Point* &*ray_origin*, **const** *Point* &*ray_direction*, *FloatingPoint* *max_dist*, *Point* **intersect_point*, *FloatingPoint* **intersect_dist*) **const**
Raycasting accessors.

Protected Attributes

Point **size_**

Class Cylinder

- Defined in *File objects.h*

Inheritance Relationships

Base Type

- `public voxblox::Object` (*Class Object*)

Class Documentation

class Cylinder : **public** voxblox::*Object*

Public Functions

Cylinder(**const** *Point* &*center*, *FloatingPoint* *radius*, *FloatingPoint* *height*)

Cylinder(**const** *Point* &*center*, *FloatingPoint* *radius*, *FloatingPoint* *height*, **const** *Color* &*color*)

virtual *FloatingPoint* **getDistanceToPoint**(**const** *Point* &*point*) **const**
Map-building accessors.

virtual bool **getRayIntersection**(**const** *Point* &*ray_origin*, **const** *Point* &*ray_direction*, *FloatingPoint* *max_dist*, *Point* **intersect_point*, *FloatingPoint* **intersect_dist*) **const**
Raycasting accessors.

Protected Attributes

FloatingPoint **radius_**

FloatingPoint **height_**

Class EsdfIntegrator

- Defined in *File esdf_integrator.h*

Nested Relationships

Nested Types

- *Struct EsdfIntegrator::Config*

Class Documentation

class EsdfIntegrator

Builds an ESDF layer out of a given TSDF layer.

For a description of this algorithm, please see: https://arxiv.org/abs/1611.03631

Public Functions

EsdfIntegrator(**const** *Config* &*config*, *Layer*<*TsdfVoxel*> **tsdf_layer*, *Layer*<*EsdfVoxel*> **esdf_layer*)

void **addNewRobotPosition**(**const** *Point* &*position*)

Used for planning - allocates sphere around as observed but occupied, and clears space in a smaller sphere around current position.

Points added this way are marked as "hallucinated," and can subsequently be cleared based on this.

void **updateFromTsdfLayerBatch**()

Update from a TSDF layer in batch, clearing the current ESDF layer in the process.

void **updateFromTsdfLayer**(bool *clear_updated_flag*)

Incrementally update from the TSDF layer, optionally clearing the updated flag of all changed TSDF voxels.

void **updateFromTsdfBlocks**(**const** *BlockIndexList* &*tsdf_blocks*, bool *incremental* = false)

Short-cut for pushing neighbors (i.e., incremental update) by default.

Not necessary in batch.

void **processRaiseSet**()

For incremental updates, the raise set contains all fixed voxels whose distances have INCREASED since last iteration.

This means that all voxels that have these voxels as parents need to be invalidated and assigned new values. The raise set is always empty in batch operations.

void **processOpenSet**()
The core of ESDF updates: processes a queue of voxels to get the minimum possible distance value in each voxel, by checking the values of its neighbors and distance to neighbors.

The update is done once the open set is empty.

bool **updateVoxelFromNeighbors**(**const** *GlobalIndex* &*global_index*)
For new voxels, etc.

update its value from its neighbors. Sort of the inverse of what the open set does (pushes the value of voxels *TO* its neighbors). Used mostly for adding new voxels in.

bool **isFixed**(*FloatingPoint dist_m*) **const**

void **clear**()
Clears the state of the integrator, in case robot pose clearance is used.

float **getEsdfMaxDistance**() **const**
Update some specific settings.

void **setEsdfMaxDistance**(float *max_distance*)

bool **getFullEuclidean**() **const**

void **setFullEuclidean**(bool *full_euclidean*)

Protected Attributes

Config **config_**

Layer<*TsdfVoxel*> ***tsdf_layer_**

Layer<*EsdfVoxel*> ***esdf_layer_**

BucketQueue<*GlobalIndex*> **open_**
Open Queue for incremental updates.

Contains global voxel indices for the ESDF layer.

AlignedQueue<*GlobalIndex*> **raise_**
Raise set for updates; these are values that used to be in the fixed frontier and now have a higher value, or their children which need to have their values invalidated.

size_t **voxels_per_side_**

FloatingPoint **voxel_size_**

IndexSet **updated_blocks_**

struct Config

Public Members

bool **full_euclidean_distance** = false
Whether to use full euclidean distance (true) or quasi-euclidean (false).

Full euclidean is slightly more accurate (up to 8% in the worst case) but slower.

FloatingPoint **max_distance_m** = 2.0
Maximum distance to calculate the actual distance to.

Any values above this will be set to default_distance_m.

FloatingPoint **min_distance_m** = 0.2
Should mirror (or be smaller than) truncation distance in tsdf integrator.

FloatingPoint **default_distance_m** = 2.0
Default distance set for unknown values and values > max_distance_m.

FloatingPoint **min_diff_m** = 0.001
For cheaper but less accurate map updates: the minimum difference in a voxel distance, before the change is propagated.

float **min_weight** = 1e-6
Minimum weight to consider a TSDF value seen at.

int **num_buckets** = 20
Number of buckets for the bucketed priority queue.

bool **multi_queue** = false
Whether to push stuff to the open queue multiple times, with updated distances.

bool **add_occupied_crust** = false
Whether to add an outside layer of occupied voxels.

Basically just sets all unknown voxels in the allocated blocks to occupied. Try to only use this for batch processing, otherwise look into addNewRobotPosition below, which uses clear spheres.

FloatingPoint **clear_sphere_radius** = 1.5
For marking unknown space around a robot as free or occupied, these are the radiuses used around each robot position.

FloatingPoint **occupied_sphere_radius** = 5.0

Class EsdfMap

- Defined in *File esdf_map.h*

Nested Relationships

Nested Types

- *Struct EsdfMap::Config*

Class Documentation

class EsdfMap
Map holding a Euclidean Signed Distance Field *Layer*.

Contains functions for interacting with the layer and getting gradient and distance information.

Public Types

typedef std::shared_ptr<*EsdfMap*> **Ptr**

using EigenDStride = Eigen::Stride<Eigen::Dynamic, Eigen::Dynamic>

using EigenDRef = Eigen::Ref<MatrixType, 0, *EigenDStride*>

Public Functions

EsdfMap(**const** *Config* &*config*)

EsdfMap(**const** *Layer*<*EsdfVoxel*> &*layer*)
Creates a new *EsdfMap* based on a COPY of this layer.

EsdfMap(*Layer*<*EsdfVoxel*>::Ptr *layer*)
Creates a new *EsdfMap* that contains this layer.

virtual ~EsdfMap()

Layer<*EsdfVoxel*> ***getEsdfLayerPtr**()

const *Layer*<*EsdfVoxel*> &**getEsdfLayer**() **const**

FloatingPoint **block_size**() **const**

FloatingPoint **voxel_size**() **const**

bool **getDistanceAtPosition**(**const** Eigen::Vector3d &*position*, double **distance*) **const**
Specific accessor functions for esdf maps.

Returns true if the point exists in the map AND is observed. These accessors use Vector3d and doubles explicitly rather than FloatingPoint to have a standard, cast-free interface to planning functions.

bool **getDistanceAtPosition**(**const** Eigen::Vector3d &*position*, bool *interpolate*, double **distance*) **const**

bool **getDistanceAndGradientAtPosition**(**const** Eigen::Vector3d &*position*, double **distance*, Eigen::Vector3d **gradient*) **const**

bool **getDistanceAndGradientAtPosition**(**const** Eigen::Vector3d &*position*, bool *interpolate*, double **distance*, Eigen::Vector3d **gradient*) **const**

bool **isObserved**(**const** Eigen::Vector3d &*position*) **const**

void **batchGetDistanceAtPosition**(*EigenDRef*<**const** Eigen::Matrix<double, 3, Eigen::Dynamic>> &*positions*, Eigen::Ref<Eigen::VectorXd> *distances*, Eigen::Ref<Eigen::VectorXi> *observed*) **const**

void **batchGetDistanceAndGradientAtPosition**(*EigenDRef*<**const** Eigen::Matrix<double, 3, Eigen::Dynamic>> &*positions*, Eigen::Ref<Eigen::VectorXd> *distances*, *EigenDRef*<Eigen::Matrix<double, 3, Eigen::Dynamic>> &*gradients*, Eigen::Ref<Eigen::VectorXi> *observed*) **const**

void **batchIsObserved**(*EigenDRef*<**const** Eigen::Matrix<double, 3, Eigen::Dynamic>> &*positions*, Eigen::Ref<Eigen::VectorXi> *observed*) **const**

unsigned int **coordPlaneSliceGetCount**(unsigned int *free_plane_index*, double *free_plane_val*) **const**

unsigned int **coordPlaneSliceGetDistance**(unsigned int *free_plane_index*, double *free_plane_val*, *EigenDRef*<Eigen::Matrix<double, 3, Eigen::Dynamic>> &*positions*, Eigen::Ref<Eigen::VectorXd> *distances*, unsigned int *max_points*) **const**

Extract all voxels on a slice plane that is parallel to one of the axis-aligned planes.

free_plane_index specifies the free coordinate (zero-based; x, y, z order) free_plane_val specifies the plane intercept coordinate along that axis

Protected Attributes

FloatingPoint **block_size_**

Layer<*EsdfVoxel*>::Ptr **esdf_layer_**

Interpolator<*EsdfVoxel*> **interpolator_**

struct Config

Public Members

FloatingPoint **esdf_voxel_size** = 0.2

size_t **esdf_voxels_per_side** = 16u

Class EsdfOccIntegrator

- Defined in *File esdf_occ_integrator.h*

Nested Relationships

Nested Types

- *Struct EsdfOccIntegrator::Config*

Class Documentation

class EsdfOccIntegrator

Builds an ESDF layer out of a given occupancy layer.

Public Functions

EsdfOccIntegrator(**const** *Config* &*config*, *Layer*<*OccupancyVoxel*> **occ_layer*, *Layer*<*EsdfVoxel*> **esdf_layer*)

void **updateFromOccLayerBatch**()
Fixed is overloaded as occupied in this case.

Only batch operations are currently supported for the occupancy map.

void **updateFromOccBlocks**(**const** *BlockIndexList* &*occ_blocks*)

void **processOpenSet**()

void **getNeighborsAndDistances**(**const** *BlockIndex* &*block_index*, **const** *VoxelIndex* &*voxel_index*, AlignedVector<*VoxelKey*> **neighbors*, AlignedVector<float> **distances*, AlignedVector<Eigen::Vector3i> **directions*) **const**
Uses 26-connectivity and quasi-Euclidean distances.

Directions is the direction that the neighbor voxel lives in. If you need the direction FROM the neighbor voxel TO the current voxel, take negative of the given direction.

void **getNeighbor**(**const** *BlockIndex* &*block_index*, **const** *VoxelIndex* &*voxel_index*, **const** Eigen::Vector3i &*direction*, *BlockIndex* **neighbor_block_index*, *VoxelIndex* **neighbor_voxel_index*) **const**

Protected Attributes

Config **config_**

Layer<*OccupancyVoxel*> ***occ_layer_**

Layer<*EsdfVoxel*> ***esdf_layer_**

BucketQueue<*VoxelKey*> **open_**
Open Queue for incremental updates.

Contains global voxel indices for the ESDF layer.

AlignedQueue<*VoxelKey*> **raise_**
Raise set for updates; these are values that used to be in the fixed frontier and now have a higher value, or their children which need to have their values invalidated.

size_t **esdf_voxels_per_side_**

FloatingPoint **esdf_voxel_size_**

struct Config

Public Members

FloatingPoint **max_distance_m** = 2.0
Maximum distance to calculate the actual distance to.

Any values above this will be set to default_distance_m.

FloatingPoint **default_distance_m** = 2.0
Default distance set for unknown values and values > max_distance_m.

int **num_buckets** = 20
Number of buckets for the bucketed priority queue.

Class EsdfServer

- Defined in *File esdf_server.h*

Inheritance Relationships

Base Type

- `public voxblox::TsdfServer` (*Class TsdfServer*)

Class Documentation

class EsdfServer : **public** voxblox::*TsdfServer*

Public Functions

EsdfServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*)

EsdfServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*, **const** *EsdfMap*::*Config* &*esdf_config*, **const** *EsdfIntegrator*::*Config* &*esdf_integrator_config*, **const** *TsdfMap*::*Config* &*tsdf_config*, **const** *TsdfIntegratorBase*::*Config* &*tsdf_integrator_config*)

virtual ~EsdfServer()

bool **generateEsdfCallback**(std_srvs::Empty::Request &*request*, std_srvs::Empty::Response &*response*)

void **publishAllUpdatedEsdfVoxels**()

virtual void **publishSlices**()

void **publishTraversable**()

virtual void **updateMesh**()
Incremental update.

virtual void **publishPointclouds**()

virtual void **newPoseCallback**(**const** *Transformation* &*T_G_C*)

virtual void **publishMap**(**const** bool *reset_remote_map* = false)

virtual bool **saveMap**(**const** std::string &*file_path*)

virtual bool **loadMap**(**const** std::string &*file_path*)

void **updateEsdf**()
Call updateMesh if you want everything updated; call this specifically if you don't want the mesh or visualization.

void **updateEsdfBatch**(bool *full_euclidean* = false)

void **esdfMapCallback**(**const** voxblox_msgs::Layer &*layer_msg*)

std::shared_ptr<*EsdfMap*> **getEsdfMapPtr**()

std::shared_ptr<**const** *EsdfMap*> **getEsdfMapPtr**() **const**

bool **getClearSphere**() **const**

void **setClearSphere**(bool *clear_sphere_for_planning*)

float **getEsdfMaxDistance**() **const**

void **setEsdfMaxDistance**(float *max_distance*)

float **getTraversabilityRadius**() **const**

void **setTraversabilityRadius**(float *traversability_radius*)

void **disableIncrementalUpdate**()
These are for enabling or disabling incremental update of the ESDF.

Use carefully.

void **enableIncrementalUpdate**()

virtual void **clear**()
CLEARS THE ENTIRE MAP!

Protected Functions

void **setupRos**()
Sets up publishing and subscribing.

Should only be called from constructor.

Protected Attributes

ros::Publisher **esdf_pointcloud_pub_**

ros::Publisher **esdf_slice_pub_**

ros::Publisher **traversable_pub_**

ros::Publisher **esdf_map_pub_**
Publish the complete map for other nodes to consume.

ros::Subscriber **esdf_map_sub_**
Subscriber to subscribe to another node generating the map.

ros::ServiceServer **generate_esdf_srv_**
Services.

bool **clear_sphere_for_planning_**

bool **publish_esdf_map_**

bool **publish_traversable_**

float **traversability_radius_**

bool **incremental_update_**

std::shared_ptr<*EsdfMap*> **esdf_map_**

std::unique_ptr<*EsdfIntegrator*> **esdf_integrator_**

Class FastTsdfIntegrator

- Defined in *File tsdf_integrator.h*

Inheritance Relationships

Base Type

- `public voxblox::TsdfIntegratorBase` (*Class TsdfIntegratorBase*)

Class Documentation

class FastTsdfIntegrator : **public** voxblox::*TsdfIntegratorBase*

An integrator that prioritizes speed over everything else.

Rays are cast from the pointcloud to the sensor origin. If a ray intersects max_consecutive_ray_collisions voxels in a row that have already been updated by other rays from the same cloud, it is terminated early. This results in a large reduction in the number of freespace updates and greatly improves runtime while ensuring all voxels receive at least a minimum number of updates. Speed is further enhanced through limiting the number of rays cast from each voxel as set by start_voxel_subsampling_factor and use of the *ApproxHashSet*. Up to an order of magnitude faster then the other integrators for small voxels.

Public Functions

FastTsdfIntegrator(**const** Config &*config*, *Layer*<*TsdfVoxel*> **layer*)

void **integrateFunction**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, **const** bool *freespace_points*, *ThreadSafeIndex* **index_getter*)

void **integratePointCloud**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, **const** bool *freespace_points* = false)

Integrates the given point infomation into the TSDF.

NOT thread safe.

Parameters

- `freespace_points`: if true points will only be integrated up to the truncation distance. Used when we are given a minimum distance to a point, rather then exact distance. This is useful for clearing out free space.

Class GrayscaleColorMap

- Defined in *File color_maps.h*

Inheritance Relationships

Base Type

- `public voxblox::ColorMap` (*Class ColorMap*)

Class Documentation

class GrayscaleColorMap : **public** voxblox::*ColorMap*

Public Functions

virtual *Color* **colorLookup**(float *value*) **const**

Class ICP

- Defined in *File icp.h*

Nested Relationships

Nested Types

- *Struct ICP::Config*

Class Documentation

class ICP

A class that performs point matching in an *ICP* like fashion to align a pointcloud with the existing TSDF information.

Note the process is slightly different to traditional *ICP*: 1) A "mini batch" of points is selected and the transform that gives the minimum least squares error for their alignment is found.

2) The "information" contained in this alignment is estimated and used to fuse this refined transform with the initial guess.

3) The process is repeated with a new mini batch of points until all points in the pointcloud have been used.

This scheme does not really iterate and uses the TSDF distances instead of building a kdtree. Both choices limit the capture region and so so assumes the initial guess is reasonably accurate. However, these limitations allow efficient correspondence estimation and parallelization, allowing efficient real time performance.

Public Functions

ICP(**const** *Config* &*config*)

A normal member taking two arguments and returning an integer value.

Parameters

- `config`: struct holding all relevant *ICP* parameters.

size_t **runICP**(**const** *Layer*<*TsdfVoxel*> &*tsdf_layer*, **const** *Pointcloud* &*points*, **const** *Transformation* &*inital_T_tsdf_sensor*, *Transformation* **refined_T_tsdf_sensor*, **const** unsigned *seed* = std::chrono::system_clock::now().time_since_epoch().count())

Runs the *ICP* method to align the points with the tsdf_layer.

Return the number of mini batches that were successful.

bool **refiningRollPitch**()

struct Config

Contains all the information needed to setup the *ICP* class.

Public Members

bool **refine_roll_pitch** = false

int **mini_batch_size** = 20

Number of points used in each alignment step.

To allow simple threading the *ICP* process is split up into a large number of separate alignments performed on small pointclouds. This parameter dictates how many points are used in each "mini batch". The result are then combined weighting them by an estimate of the information gained by the alignment.

FloatingPoint **min_match_ratio** = 0.8

Ratio of points that must lie within the truncation distance of an allocated voxel.

FloatingPoint **subsample_keep_ratio** = 0.5

Ratio of points used in the *ICP* matching.

FloatingPoint **inital_translation_weighting** = 100.0

Weighting applied to the translational component of the initial guess.

Very roughly corresponds to the inverse covariance of the initial guess multiplied by the variance in a measured points accuracy.

FloatingPoint **inital_rotation_weighting** = 100.0

Weighting applied to the rotational component of the initial guess.

See inital_translation_weighting for further details

size_t **num_threads** = std::thread::hardware_concurrency()

Class IntensityIntegrator

- Defined in *File intensity_integrator.h*

Class Documentation

class IntensityIntegrator

Integrates intensities from a set of bearing vectors (i.e., an intensity image, such as a thermal image) by projecting them onto the TSDF surface and coloring near the surface crossing.

Public Functions

IntensityIntegrator(**const** *Layer*<*TsdfVoxel*> &*tsdf_layer*, *Layer*<*IntensityVoxel*> **intensity_layer*)

void **setMaxDistance**(**const** *FloatingPoint max_distance*)
Set the max distance for projecting into the TSDF layer.

FloatingPoint **getMaxDistance**() **const**

void **addIntensityBearingVectors**(**const** *Point* &*origin*, **const** *Pointcloud* &*bearing_vectors*, **const** std::vector<float> &*intensities*)
Integrates intensities into the intensity layer by projecting normalized bearing vectors (in the WORLD coordinate frame) from the origin (also in the world coordinate frame) into the TSDF layer, and then setting the intensities near the surface boundaries.

Class IntensityServer

- Defined in *File intensity_server.h*

Inheritance Relationships

Base Type

- `public voxblox::TsdfServer` (*Class TsdfServer*)

Class Documentation

class IntensityServer : **public** voxblox::*TsdfServer*

Public Functions

IntensityServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*)

virtual ~IntensityServer()

virtual void **updateMesh**()
Incremental update.

virtual void **publishPointclouds**()

void **intensityImageCallback**(**const** sensor_msgs::ImageConstPtr &*image*)

Protected Attributes

ros::Subscriber **intensity_image_sub_**
Subscriber for intensity images.

ros::Publisher **intensity_pointcloud_pub_**

ros::Publisher **intensity_mesh_pub_**

double **focal_length_px_**
Parameters of the incoming UNDISTORTED intensity images.

int **subsample_factor_**
How much to subsample the image by (not proper downsampling, just subsampling).

std::shared_ptr<*Layer*<*IntensityVoxel*>> **intensity_layer_**

std::unique_ptr<*IntensityIntegrator*> **intensity_integrator_**

std::shared_ptr<*ColorMap*> **color_map_**

Class InteractiveSlider

- Defined in *File interactive_slider.h*

Class Documentation

class InteractiveSlider
InteractiveSlider class which can be used for visualizing voxel map slices.

Public Functions

InteractiveSlider(**const** std::string &*slider_name*, **const** std::function<void(**const** double &slice_level)> &*slider_callback*, **const** *Point* &*initial_position*, **const** unsigned int *free_plane_index*, **const** float *marker_scale_meters*)

virtual ~InteractiveSlider()

Template Class Interpolator

- Defined in *File interpolator.h*

Class Documentation

template <typename *VoxelType*>
class Interpolator
Interpolates voxels to give distances and gradients.

Public Types

typedef std::shared_ptr<*Interpolator*> **Ptr**

Public Functions

Interpolator(**const** *Layer*<VoxelType> **layer*)

bool **getGradient**(**const** *Point* &*pos*, *Point* **grad*, **const** bool *interpolate* = false) **const**

bool **getDistance**(**const** *Point* &*pos*, *FloatingPoint* **distance*, bool *interpolate* = false) **const**

bool **getVoxel**(**const** *Point* &*pos*, VoxelType **voxel*, bool *interpolate* = false) **const**

bool **getAdaptiveDistanceAndGradient**(**const** *Point* &*pos*, *FloatingPoint* **distance*, *Point* **grad*) **const**

This tries to use whatever information is available to interpolate the distance and gradient if only one side is available, for instance, this will still estimate a 1-sided gradient.

Should give the same results as *getGradient()* and *getDistance()* if all neighbors are filled.

bool **getNearestDistanceAndWeight**(**const** *Point* &*pos*, *FloatingPoint* **distance*, float **weight*) **const**

Without interpolation.

Class InverseGrayscaleColorMap

- Defined in *File color_maps.h*

Inheritance Relationships

Base Type

- `public voxblox::ColorMap` (*Class ColorMap*)

Class Documentation

class InverseGrayscaleColorMap : **public** voxblox::*ColorMap*

Public Functions

virtual *Color* **colorLookup**(float *value*) **const**

Class InverseRainbowColorMap

- Defined in *File color_maps.h*

Inheritance Relationships

Base Type

- `public voxblox::ColorMap` (*Class ColorMap*)

Class Documentation

class InverseRainbowColorMap : **public** voxblox::*ColorMap*

Public Functions

virtual *Color* **colorLookup**(float *value*) **const**

Class PlyWriter

- Defined in *File ply_writer.h*

Class Documentation

class PlyWriter

Writes a mesh to a .ply file.

For reference on the format, see: http://paulbourke.net/dataformats/ply/

Public Functions

PlyWriter(**const** std::string &*filename*)

virtual ~PlyWriter()

void **addVerticesWithProperties**(size_t *num_vertices*, bool *has_color*)

bool **writeHeader**()

bool **writeVertex**(**const** *Point* &*coord*)

bool **writeVertex**(**const** *Point* &*coord*, **const** *Color* &*rgb*)

void **closeFile**()

Class IronbowColorMap

- Defined in *File color_maps.h*

Inheritance Relationships

Base Type

- `public voxblox::ColorMap` (*Class ColorMap*)

Class Documentation

class IronbowColorMap : **public** voxblox::*ColorMap*

Public Functions

IronbowColorMap()

virtual *Color* **colorLookup**(float *value*) **const**

Protected Attributes

std::vector<*Color*> **palette_colors_**

float **increment_**

Template Class Layer

- Defined in *File layer.h*

Class Documentation

template <typename *VoxelType*>
class Layer

A 3D information layer, containing data of type VoxelType stored in blocks.

This class contains functions for manipulating and accessing these blocks.

Public Types

enum BlockMergingStrategy

Values:

kProhibit

kReplace

kDiscard

kMerge

typedef std::shared_ptr<*Layer*> **Ptr**

typedef *Block*<VoxelType> **BlockType**

typedef *AnyIndexHashMapType*<**typename** *BlockType*::Ptr>::type **BlockHashMap**

typedef std::pair<*BlockIndex*, **typename** *BlockType*::Ptr> **BlockMapPair**

Public Functions

Layer(*FloatingPoint voxel_size*, size_t *voxels_per_side*)

Layer(**const** LayerProto &*proto*)

Create the layer from protobuf layer header.

Layer(**const** *Layer* &*other*)

Deep copy constructor.

virtual ~Layer()

const *BlockType* &**getBlockByIndex**(**const** *BlockIndex* &*index*) **const**

BlockType &**getBlockByIndex**(**const** *BlockIndex* &*index*)

BlockType::ConstPtr **getBlockPtrByIndex**(**const** *BlockIndex* &*index*) **const**

BlockType::Ptr **getBlockPtrByIndex**(**const** *BlockIndex* &*index*)

BlockType::Ptr **allocateBlockPtrByIndex**(**const** *BlockIndex* &*index*)
Gets a block by the block index it if already exists, otherwise allocates a new one.

BlockType::ConstPtr **getBlockPtrByCoordinates**(**const** *Point* &*coords*) **const**

BlockType::Ptr **getBlockPtrByCoordinates**(**const** *Point* &*coords*)

BlockType::Ptr **allocateBlockPtrByCoordinates**(**const** *Point* &*coords*)
Gets a block by the coordinates it if already exists, otherwise allocates a new one.

BlockIndex **computeBlockIndexFromCoordinates**(**const** *Point* &*coords*) **const**
IMPORTANT NOTE: Due the limited accuracy of the FloatingPoint type, this function doesn't always compute the correct block index for coordinates near the block boundaries.

BlockType::Ptr **allocateNewBlock**(**const** *BlockIndex* &*index*)

BlockType::Ptr **allocateNewBlockByCoordinates**(**const** *Point* &*coords*)

void **insertBlock**(**const** std::pair<**const** *BlockIndex*, **typename** *Block*<VoxelType>::Ptr> &*block_pair*)

void **removeBlock**(**const** *BlockIndex* &*index*)

void **removeAllBlocks**()

void **removeBlockByCoordinates**(**const** *Point* &*coords*)

void **removeDistantBlocks**(**const** *Point* &*center*, **const** double *max_distance*)

void **getAllAllocatedBlocks**(*BlockIndexList* **blocks*) **const**

void **getAllUpdatedBlocks**(*BlockIndexList* **blocks*) **const**

size_t **getNumberOfAllocatedBlocks**() **const**

bool **hasBlock**(**const** *BlockIndex* &*block_index*) **const**

const VoxelType ***getVoxelPtrByGlobalIndex**(**const** *GlobalIndex* &*global_voxel_index*) **const**
Get a pointer to the voxel if its corresponding block is allocated and a nullptr otherwise.

VoxelType ***getVoxelPtrByGlobalIndex**(**const** *GlobalIndex* &*global_voxel_index*)

const VoxelType ***getVoxelPtrByCoordinates**(**const** *Point* &*coords*) **const**

VoxelType ***getVoxelPtrByCoordinates**(**const** *Point* &*coords*)

FloatingPoint **block_size**() **const**

FloatingPoint **block_size_inv**() **const**

FloatingPoint **voxel_size**() **const**

FloatingPoint **voxel_size_inv**() **const**

size_t **voxels_per_side**() **const**

FloatingPoint **voxels_per_side_inv**() **const**

void **getProto**(LayerProto **proto*) **const**

bool **isCompatible**(**const** LayerProto &*layer_proto*) **const**

bool **isCompatible**(**const** BlockProto &*layer_proto*) **const**

bool **saveToFile**(**const** std::string &*file_path*, bool *clear_file* = true) **const**

bool **saveSubsetToFile**(**const** std::string &*file_path*, *BlockIndexList* *blocks_to_include*, bool *include_all_blocks*, bool *clear_file* = true) **const**

bool **addBlockFromProto**(**const** BlockProto &*block_proto*, *BlockMergingStrategy* *strategy*)

size_t **getMemorySize**() **const**

Protected Functions

std::string **getType**() **const**

Protected Attributes

FloatingPoint **voxel_size_**

size_t **voxels_per_side_**

FloatingPoint **block_size_**

FloatingPoint **voxel_size_inv_**

FloatingPoint **block_size_inv_**

FloatingPoint **voxels_per_side_inv_**

BlockHashMap **block_map_**

Class MarchingCubes

- Defined in *File marching_cubes.h*

Class Documentation

class MarchingCubes

Performs the marching cubes algorithm to generate a mesh layer from a TSDF.

Implementation taken from Open Chisel https://github.com/personalrobotics/OpenChisel

Public Functions

MarchingCubes()

virtual ~MarchingCubes()

Public Static Functions

static void **meshCube**(**const** Eigen::Matrix<*FloatingPoint*, 3, 8> &*vertex_coordinates*, **const** Eigen::Matrix<*FloatingPoint*, 8, 1> &*vertex_sdf*, *TriangleVector* **triangles*)

static void **meshCube**(**const** Eigen::Matrix<*FloatingPoint*, 3, 8> &*vertex_coords*, **const** Eigen::Matrix<*FloatingPoint*, 8, 1> &*vertex_sdf*, *VertexIndex* **next_index*, *Mesh* **mesh*)

static int **calculateVertexConfiguration**(**const** Eigen::Matrix<*FloatingPoint*, 8, 1> &*vertex_sdf*)

static void **interpolateEdgeVertices**(**const** Eigen::Matrix<*FloatingPoint*, 3, 8> &*vertex_coords*, **const** Eigen::Matrix<*FloatingPoint*, 8, 1> &*vertex_sdf*, Eigen::Matrix<*FloatingPoint*, 3, 12> **edge_coords*)

static *Point* **interpolateVertex**(**const** *Point* &*vertex1*, **const** *Point* &*vertex2*, float *sdf1*, float *sdf2*)

Performs linear interpolation on two cube corners to find the approximate zero crossing (surface) value.

Public Static Attributes

int **kTriangleTable**[256][16]

int **kEdgeIndexPairs**[12][2]

Class MergedTsdfIntegrator

- Defined in *File tsdf_integrator.h*

Inheritance Relationships

Base Type

- `public voxblox::TsdfIntegratorBase` (*Class TsdfIntegratorBase*)

Class Documentation

class MergedTsdfIntegrator : **public** voxblox::*TsdfIntegratorBase*

Uses ray bundling to improve integration speed, points which lie in the same voxel are "merged" into a single point.

Raycasting and updating then proceeds as normal. Fast for large voxels, with minimal loss of information.

Public Functions

MergedTsdfIntegrator(**const** Config &*config*, *Layer*<*TsdfVoxel*> **layer*)

void **integratePointCloud**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, **const** bool *freespace_points* = false)

Integrates the given point infomation into the TSDF.

NOT thread safe.

Parameters

- `freespace_points`: if true points will only be integrated up to the truncation distance. Used when we are given a minimum distance to a point, rather then exact distance. This is useful for clearing out free space.

Protected Functions

void **bundleRays**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** bool *freespace_points*, *ThreadSafeIndex* **index_getter*, *LongIndexHashMapType*<AlignedVector<size_t>>::type **voxel_map*, *LongIndexHashMapType*<AlignedVector<size_t>>::type **clear_map*)

void **integrateVoxel**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, bool *enable_anti_grazing*, bool *clearing_ray*, **const** std::pair<*GlobalIndex*, AlignedVector<size_t>> &*kv*, **const** *LongIndexHashMapType*<AlignedVector<size_t>>::type &*voxel_map*)

void **integrateVoxels**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, bool *enable_anti_grazing*, bool *clearing_ray*, **const** *LongIndexHashMapType*<AlignedVector<size_t>>::type &*voxel_map*, **const** *LongIndexHashMapType*<AlignedVector<size_t>>::type &*clear_map*, size_t *thread_idx*)

void **integrateRays**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, bool *enable_anti_grazing*, bool *clearing_ray*, **const** *LongIndexHashMapType*<AlignedVector<size_t>>::type &*voxel_map*, **const** *LongIndexHashMapType*<AlignedVector<size_t>>::type &*clear_map*)

Template Class MeshIntegrator

- Defined in *File mesh_integrator.h*

Class Documentation

template <**typename** VoxelType>
class MeshIntegrator

Integrates a TSDF layer to incrementally update a mesh layer using marching cubes.

Public Functions

void **initFromSdfLayer**(**const** *Layer*<VoxelType> &*sdf_layer*)

MeshIntegrator(**const** *MeshIntegratorConfig* &*config*, *Layer*<VoxelType> **sdf_layer*, *MeshLayer* **mesh_layer*)

Use this constructor in case you would like to modify the layer during mesh extraction, i.e.

modify the updated flag.

MeshIntegrator(**const** *MeshIntegratorConfig* &*config*, **const** *Layer*<VoxelType> &*sdf_layer*, *MeshLayer* **mesh_layer*)

This constructor will not allow you to modify the layer, i.e.

clear the updated flag.

void **generateMesh**(bool *only_mesh_updated_blocks*, bool *clear_updated_flag*)

Generates mesh from the tsdf layer.

void **generateMeshBlocksFunction**(**const** *BlockIndexList* &*all_tsdf_blocks*, bool *clear_updated_flag*, *ThreadSafeIndex* **index_getter*)

void **extractBlockMesh**(**typename** *Block*<VoxelType>::ConstPtr *block*, *Mesh*::*Ptr mesh*)

virtual void **updateMeshForBlock**(**const** *BlockIndex* &*block_index*)

void **extractMeshInsideBlock**(**const** *Block*<VoxelType> &*block*, **const** *VoxelIndex* &*index*, **const** *Point* &*coords*, *VertexIndex* **next_mesh_index*, *Mesh* **mesh*)

void **extractMeshOnBorder**(**const** *Block*<VoxelType> &*block*, **const** *VoxelIndex* &*index*, **const** *Point* &*coords*, *VertexIndex* **next_mesh_index*, *Mesh* **mesh*)

void **updateMeshColor**(**const** *Block*<VoxelType> &*block*, *Mesh* **mesh*)

Protected Attributes

MeshIntegratorConfig **config_**

Layer<VoxelType> ***sdf_layer_mutable_**

Having both a const and a mutable pointer to the layer allows this integrator to work both with a const layer (in case you don't want to clear the updated flag) and mutable layer (in case you do want to clear the updated flag).

const *Layer*<VoxelType> ***sdf_layer_const_**

MeshLayer ***mesh_layer_**

FloatingPoint **voxel_size_**

size_t **voxels_per_side_**

FloatingPoint **block_size_**

FloatingPoint **voxel_size_inv_**

FloatingPoint **voxels_per_side_inv_**

FloatingPoint **block_size_inv_**

Eigen::Matrix<int, 3, 8> **cube_index_offsets_**

Class MeshLayer

- Defined in *File mesh_layer.h*

Class Documentation

class MeshLayer

A special type of layer just for containing the mesh.

Same general interface as a layer of blocks, but only contains a single thing, not a set of voxels.

Public Types

typedef std::shared_ptr<*MeshLayer*> **Ptr**

typedef std::shared_ptr<**const** *MeshLayer*> **ConstPtr**

typedef *AnyIndexHashMapType*<*Mesh*::*Ptr*>::type **MeshMap**

Public Functions

MeshLayer(*FloatingPoint block_size*)

virtual ~MeshLayer()

const *Mesh* &**getMeshByIndex**(**const** *BlockIndex* &*index*) **const**

Mesh &**getMeshByIndex**(**const** *BlockIndex* &*index*)

Mesh::*ConstPtr* **getMeshPtrByIndex**(**const** *BlockIndex* &*index*) **const**

Mesh::*Ptr* **getMeshPtrByIndex**(**const** *BlockIndex* &*index*)

Mesh::*Ptr* **allocateMeshPtrByIndex**(**const** *BlockIndex* &*index*)
Gets a mesh by the mesh index it if already exists, otherwise allocates a new one.

Mesh::*ConstPtr* **getMeshPtrByCoordinates**(**const** *Point* &*coords*) **const**

Mesh::*Ptr* **getMeshPtrByCoordinates**(**const** *Point* &*coords*)

Mesh::*Ptr* **allocateMeshPtrByCoordinates**(**const** *Point* &*coords*)
Gets a mesh by the coordinates it if already exists, otherwise allocates a new one.

BlockIndex **computeBlockIndexFromCoordinates**(**const** *Point* &*coords*) **const**
Coord to mesh index.

Mesh::*Ptr* **allocateNewBlock**(**const** *BlockIndex* &*index*)

Mesh::*Ptr* **allocateNewBlockByCoordinates**(**const** *Point* &*coords*)

void **removeMesh**(**const** *BlockIndex* &*index*)

void **removeMeshByCoordinates**(**const** *Point* &*coords*)

void **clearDistantMesh**(**const** *Point* &*center*, **const** double *max_distance*)

void **getAllAllocatedMeshes**(*BlockIndexList* **meshes*) **const**

void **getAllUpdatedMeshes**(*BlockIndexList* **meshes*) **const**

void **getMesh**(*Mesh* *combined_mesh) **const**
Get mesh from mesh layer.

NOTE: The triangles and vertices in this mesh are distinct, hence, this will not produce a connected mesh.

void voxblox::MeshLayer::getConnectedMesh(Mesh * connected_mesh, const FloatingPoint
Get a connected mesh by merging close vertices and removing triangles with zero surface area.

If you only would like to connect vertices, make sure that the proximity threhsold <<< voxel size. If you would like to simplify the mesh, chose a threshold greater or near the voxel size until you reached the level of simpliciation desired.

size_t **getNumberOfAllocatedMeshes**() **const**

void **clear**()
Deletes ALL parts of the mesh.

FloatingPoint **block_size**() **const**

FloatingPoint **block_size_inv**() **const**

Template Class Neighborhood

- Defined in *File neighbor_tools.h*

Inheritance Relationships

Base Type

- `public voxblox::NeighborhoodLookupTables` (*Class NeighborhoodLookupTables*)

Class Documentation

template <*Connectivity kConnectivity* = Connectivity::kTwentySix>
class Neighborhood : **public** voxblox::*NeighborhoodLookupTables*

Public Types

typedef Eigen::Matrix<*LongIndexElement*, 3, kConnectivity> **IndexMatrix**

Public Static Functions

static void **getFromGlobalIndex**(**const** *GlobalIndex* &*global_index*, *IndexMatrix* *neighbors)
Get the global index of all (6, 18, or 26) neighbors of the input index.

static void **getFromBlockAndVoxelIndexAndDirection**(**const** *BlockIndex* &*block_index*, **const** *VoxelIndex* &*voxel_index*, **const** *SignedIndex* &*direction*, **const** size_t *voxels_per_side*, *BlockIndex* **neighbor_block_index*, *VoxelIndex* **neighbor_voxel_index*)

Get the block idx and local voxel index of a neighbor voxel.

The neighbor voxel is defined by providing a direction. The main purpose of this function is to solve the cross-block indexing that happens when looking up neighbors at the block boundaries.

static void **getFromBlockAndVoxelIndex**(**const** *BlockIndex* &*block_index*, **const** *VoxelIndex* &*voxel_index*, **const** size_t *voxels_per_side*, AlignedVector<*VoxelKey*> **neighbors_ptr*)

Get the hierarchical indices (block idx, local voxel index) for all neighbors (6, 18, or 26 neighborhood) of a hierarcical index.

This function solves the cross-block indexing that happens when looking up neighbors at the block boundary.

static *SignedIndex* **getOffsetBetweenVoxels**(**const** *BlockIndex* &*start_block_index*, **const** *VoxelIndex* &*start_voxel_index*, **const** *BlockIndex* &*end_block_index*, **const** *VoxelIndex* &*end_voxel_index*, **const** size_t *voxels_per_side*)

Get the signed offset between the global indices of two voxels.

Class NeighborhoodLookupTables

- Defined in *File neighbor_tools.h*

Inheritance Relationships

Derived Type

- `public voxblox::Neighborhood< kConnectivity >` (*Template Class Neighborhood*)

Class Documentation

class NeighborhoodLookupTables

Subclassed by *voxblox::Neighborhood< kConnectivity >*

Public Types

typedef Eigen::Matrix<*LongIndexElement*, 3, *Connectivity*::kTwentySix> **LongIndexOffsets**

typedef Eigen::Matrix<*IndexElement*, 3, *Connectivity*::kTwentySix> **IndexOffsets**

typedef Eigen::Matrix<float, 1, *Connectivity*::kTwentySix> **Distances**

Public Static Attributes

const *Distances* **kDistances**

const *IndexOffsets* **kOffsets**

const *LongIndexOffsets* **kLongOffsets**

Class Object

- Defined in *File objects.h*

Inheritance Relationships

Derived Types

- `public voxblox::Cube` (*Class Cube*)
- `public voxblox::Cylinder` (*Class Cylinder*)
- `public voxblox::PlaneObject` (*Class PlaneObject*)
- `public voxblox::Sphere` (*Class Sphere*)

Class Documentation

class Object

Base class for simulator objects.

Each object allows an exact ground-truth sdf to be created for it.

Subclassed by *voxblox::Cube*, *voxblox::Cylinder*, *voxblox::PlaneObject*, *voxblox::Sphere*

Public Types

enum Type

Values:

kSphere = 0

kCube

kPlane

kCylinder

Public Functions

Object(**const** *Point* &*center*, *Type type*)

Object(**const** *Point* &*center*, *Type type*, **const** *Color* &*color*)

virtual ~Object()

virtual *FloatingPoint* **getDistanceToPoint**(**const** *Point* &*point*) **const** = 0
Map-building accessors.

Color **getColor**() **const**

virtual bool **getRayIntersection**(**const** *Point* &*ray_origin*, **const** *Point* &*ray_direction*, *FloatingPoint* *max_dist*, *Point* **intersect_point*, *FloatingPoint* **intersect_dist*) **const** = 0
Raycasting accessors.

Protected Attributes

Point **center_**

Type **type_**

Color **color_**

Class OccupancyIntegrator

- Defined in *File occupancy_integrator.h*

Nested Relationships

Nested Types

- *Struct OccupancyIntegrator::Config*

Class Documentation

class OccupancyIntegrator
Integrates a pointcloud into an occupancy layer by raycasting the points and updating all the voxels the rays pass through.

Public Functions

OccupancyIntegrator(**const** *Config* &*config*, *Layer*<*OccupancyVoxel*> **layer*)

void **updateOccupancyVoxel**(bool *occupied*, *OccupancyVoxel* **occ_voxel*)

void **integratePointCloud**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*)

Protected Attributes

Config **config_**

Layer<*OccupancyVoxel*> ***layer_**

FloatingPoint **voxel_size_**

size_t **voxels_per_side_**

FloatingPoint **block_size_**

float **prob_hit_log_**

float **prob_miss_log_**

float **clamp_min_log_**

float **clamp_max_log_**

float **min_occupancy_log_**

FloatingPoint **voxel_size_inv_**

FloatingPoint **voxels_per_side_inv_**

FloatingPoint **block_size_inv_**

struct Config

Public Members

float **probability_hit** = 0.65f

float **probability_miss** = 0.4f

float **threshold_min** = 0.12f

float **threshold_max** = 0.97f

float **threshold_occupancy** = 0.7f

FloatingPoint **min_ray_length_m** = 0.1

FloatingPoint **max_ray_length_m** = 5.0

Class OccupancyMap

- Defined in *File occupancy_map.h*

Nested Relationships

Nested Types

- *Struct OccupancyMap::Config*

Class Documentation

class OccupancyMap

Map holding an Occupancy *Layer*, inspired by Octomap.

Public Types

typedef std::shared_ptr<*OccupancyMap*> **Ptr**

Public Functions

OccupancyMap(**const** *Config* &*config*)

OccupancyMap(**const** *Layer*<*OccupancyVoxel*> &*layer*)

OccupancyMap(*Layer*<*OccupancyVoxel*>::Ptr *layer*)

virtual ~OccupancyMap()

Layer<*OccupancyVoxel*> ***getOccupancyLayerPtr**()

const *Layer*<*OccupancyVoxel*> &**getOccupancyLayer**() **const**

FloatingPoint **block_size**() **const**

Protected Attributes

FloatingPoint **block_size_**

Layer<*OccupancyVoxel*>::Ptr **occupancy_layer_**

struct Config

Public Members

FloatingPoint **occupancy_voxel_size** = 0.2

size_t **occupancy_voxels_per_side** = 16u

Class Plane

- Defined in *File camera_model.h*

Class Documentation

class Plane

Represents a plane in Hesse normal form (normal + distance) for faster distance calculations to points.

Public Functions

Plane()

virtual ~Plane()

void **setFromPoints**(**const** *Point* &*p1*, **const** *Point* &*p2*, **const** *Point* &*p3*)

void **setFromDistanceNormal**(**const** *Point* &*normal*, double *distance*)

bool **isPointInside**(**const** *Point* &*point*) **const**

I guess more correctly, is the point on the correct side of the plane.

Point **normal**() **const**

double **distance**() **const**

Class PlaneObject

- Defined in *File objects.h*

Inheritance Relationships

Base Type

- `public voxblox::Object` (*Class Object*)

Class Documentation

class PlaneObject : **public** voxblox::*Object*
Requires normal being passed in to ALREADY BE NORMALIZED!!!!

Public Functions

PlaneObject(**const** *Point* &*center*, **const** *Point* &*normal*)

PlaneObject(**const** *Point* &*center*, **const** *Point* &*normal*, **const** *Color* &*color*)

virtual *FloatingPoint* **getDistanceToPoint**(**const** *Point* &*point*) **const**
Map-building accessors.

virtual bool **getRayIntersection**(**const** *Point* &*ray_origin*, **const** *Point* &*ray_direction*, *FloatingPoint* *max_dist*, *Point* **intersect_point*, *FloatingPoint* **intersect_dist*) **const**
Raycasting accessors.

Protected Attributes

Point **normal_**

Class RainbowColorMap

- Defined in *File color_maps.h*

Inheritance Relationships

Base Type

- `public voxblox::ColorMap` (*Class ColorMap*)

Class Documentation

class RainbowColorMap : **public** voxblox::*ColorMap*

Public Functions

virtual *Color* **colorLookup**(float *value*) **const**

Class RayCaster

- Defined in *File integrator_utils.h*

Class Documentation

class RayCaster

Generates the indexes of all voxels a given ray will pass through.

This class assumes PRE-SCALED coordinates, where one unit = one voxel size. The indices are also returned in this scales coordinate system, which should map to voxel indexes.

Public Functions

RayCaster(**const** *Point* &*origin*, **const** *Point* &*point_G*, **const** bool *is_clearing_ray*, **const** bool *voxel_carving_enabled*, **const** *FloatingPoint* *max_ray_length_m*, **const** *FloatingPoint* *voxel_size_inv*, **const** *FloatingPoint* *truncation_distance*, **const** bool *cast_from_origin* = true)

RayCaster(**const** *Point* &*start_scaled*, **const** *Point* &*end_scaled*)

bool **nextRayIndex**(*GlobalIndex* **ray_index*)

returns false if ray terminates at ray_index, true otherwise

Class SimpleTsdfIntegrator

- Defined in *File tsdf_integrator.h*

Inheritance Relationships

Base Type

- `public voxblox::TsdfIntegratorBase` (*Class TsdfIntegratorBase*)

Class Documentation

class SimpleTsdfIntegrator : **public** voxblox::*TsdfIntegratorBase*

Basic TSDF integrator.

Every point is raycast through all the voxels, which are updated individually. An exact but very slow approach.

Public Functions

SimpleTsdfIntegrator(**const** Config &*config*, *Layer*<*TsdfVoxel*> **layer*)

void **integratePointCloud**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, **const** bool *freespace_points* = false)

Integrates the given point infomation into the TSDF.

NOT thread safe.

Parameters

- `freespace_points`: if true points will only be integrated up to the truncation distance. Used when we are given a minimum distance to a point, rather then exact distance. This is useful for clearing out free space.

void **integrateFunction**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, **const** bool *freespace_points*, *ThreadSafeIndex* **index_getter*)

Class SimulationServer

- Defined in *File simulation_server.h*

Class Documentation

class SimulationServer

Public Functions

SimulationServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*)

SimulationServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*, **const** *EsdfMap*::*Config* &*esdf_config*, **const** *EsdfIntegrator*::*Config* &*esdf_integrator_config*, **const** *TsdfMap*::*Config* &*tsdf_config*, **const** *TsdfIntegratorBase*::*Config* &*tsdf_integrator_config*)

virtual ~SimulationServer()

void **run**()
Runs all of the below functions in the correct order:

virtual void **prepareWorld**() = 0
Creates a new world, and prepares ground truth SDF(s).

void **generateSDF**()
Generates a SDF by generating random poses and putting them into an SDF.

void **evaluate**()
Evaluate errors...

void **visualize**()
Visualize results. :)

Protected Functions

void **getServerConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

bool **generatePlausibleViewpoint**(*FloatingPoint* *min_distance*, *Point* **ray_origin*, *Point* **ray_direction*) **const**

Convenience function to generate valid viewpoints.

Protected Attributes

ros::NodeHandle **nh_**

ros::NodeHandle **nh_private_**

ros::Publisher **sim_pub_**

ros::Publisher **tsdf_gt_pub_**

ros::Publisher **esdf_gt_pub_**

ros::Publisher **tsdf_gt_mesh_pub_**

ros::Publisher **tsdf_test_pub_**

ros::Publisher **esdf_test_pub_**

ros::Publisher **tsdf_test_mesh_pub_**

ros::Publisher **view_ptcloud_pub_**

FloatingPoint **voxel_size_**

int **voxels_per_side_**

std::string **world_frame_**

bool **generate_occupancy_**

bool **visualize_**

FloatingPoint **visualization_slice_level_**

bool **generate_mesh_**

bool **incremental_**

bool **add_robot_pose_**

FloatingPoint **truncation_distance_**

FloatingPoint **esdf_max_distance_**

size_t **max_attempts_to_generate_viewpoint_**

Eigen::Vector2i **depth_camera_resolution_**

FloatingPoint **fov_h_rad_**

FloatingPoint **max_dist_**

FloatingPoint **min_dist_**

int **num_viewpoints_**

SimulationWorld **world_**

std::unique_ptr<*Layer*<*TsdfVoxel*>> **tsdf_gt_**

std::unique_ptr<*Layer*<*EsdfVoxel*>> **esdf_gt_**

std::unique_ptr<*Layer*<*TsdfVoxel*>> **tsdf_test_**

std::unique_ptr<*Layer*<*EsdfVoxel*>> **esdf_test_**

std::unique_ptr<*Layer*<*OccupancyVoxel*>> **occ_test_**

std::unique_ptr<*TsdfIntegratorBase*> **tsdf_integrator_**

std::unique_ptr<*EsdfIntegrator*> **esdf_integrator_**

std::unique_ptr<*OccupancyIntegrator*> **occ_integrator_**

std::unique_ptr<*EsdfOccIntegrator*> **esdf_occ_integrator_**

Class SimulationWorld

- Defined in *File simulation_world.h*

Class Documentation

class SimulationWorld

Public Functions

SimulationWorld()

virtual ~SimulationWorld()

void **addObject**(std::unique_ptr<*Object*> *object*)
=== Creating an environment ===

void **addGroundLevel**(*FloatingPoint* *height*)
Convenience functions for setting up bounded areas.

Ground level can also be used for ceiling. ;)

void **addPlaneBoundaries**(*FloatingPoint* *x_min*, *FloatingPoint* *x_max*, *FloatingPoint* *y_min*, *FloatingPoint* *y_max*)
Add 4 walls (infinite planes) bounding the space.

In case this is not the desired behavior, can use addObject to add walls manually one by one. If infinite walls are undesirable, then use cubes.

void **clear**()
Deletes all objects!

void **getPointcloudFromViewpoint**(**const** *Point* &*view_origin*, **const** *Point* &*view_direction*, **const** Eigen::Vector2i &*camera_res*, *FloatingPoint* *fov_h_rad*, *FloatingPoint* *max_dist*, *Pointcloud* *ptcloud*, *Colors* *colors*) **const**
=== Generating synthetic data from environment === Generates a synthetic view Assumes square pixels for ease...

Takes in FoV in radians.

void **getPointcloudFromTransform**(**const** *Transformation* &*pose*, **const** Eigen::Vector2i &*camera_res*, *FloatingPoint* *fov_h_rad*, *FloatingPoint* *max_dist*, *Pointcloud* **ptcloud*, *Colors* **colors*) **const**

void **getNoisyPointcloudFromViewpoint**(**const** *Point* &*view_origin*, **const** *Point* &*view_direction*, **const** Eigen::Vector2i &*camera_res*, *FloatingPoint* *fov_h_rad*, *FloatingPoint* *max_dist*, *FloatingPoint* *noise_sigma*, *Pointcloud* **ptcloud*, *Colors* **colors*)

Same thing as getPointcloudFromViewpoint, but also adds a noise in the *distance* of the measurement, given by noise_sigma (Gaussian noise).

No noise in the bearing.

void **getNoisyPointcloudFromTransform**(**const** *Transformation* &*pose*, **const** Eigen::Vector2i &*camera_res*, *FloatingPoint* *fov_h_rad*, *FloatingPoint* *max_dist*, *FloatingPoint* *noise_sigma*, *Pointcloud* **ptcloud*, *Colors* **colors*)

template <**typename** VoxelType>
void **generateSdfFromWorld**(*FloatingPoint* *max_dist*, *Layer*<VoxelType> **layer*) **const**

FloatingPoint **getDistanceToPoint**(**const** *Point* &*coords*, *FloatingPoint* *max_dist*) **const**

void **setBounds**(**const** *Point* &*min_bound*, **const** *Point* &*max_bound*)

Set and get the map generation and display bounds.

Point **getMinBound**() **const**

Point **getMaxBound**() **const**

template <>
void **setVoxel**(*FloatingPoint* *dist*, **const** *Color* &*color*, *TsdfVoxel* **voxel*) **const**

template <>
void **setVoxel**(*FloatingPoint* *dist*, **const** *Color*&, *EsdfVoxel* **voxel*) **const**

Class Sphere

- Defined in *File objects.h*

Inheritance Relationships

Base Type

- `public voxblox::Object` (*Class Object*)

Class Documentation

class Sphere : **public** voxblox::*Object*

Public Functions

Sphere(**const** *Point* &*center*, *FloatingPoint* *radius*)

Sphere(**const** *Point* &*center*, *FloatingPoint* *radius*, **const** *Color* &*color*)

virtual *FloatingPoint* **getDistanceToPoint**(**const** *Point* &*point*) **const**
Map-building accessors.

virtual bool **getRayIntersection**(**const** *Point* &*ray_origin*, **const** *Point* &*ray_direction*, *FloatingPoint* *max_dist*, *Point* **intersect_point*, *FloatingPoint* **intersect_dist*) **const**
Raycasting accessors.

Protected Attributes

FloatingPoint **radius_**

Template Class LayerTest

- Defined in *File layer_test_utils.h*

Class Documentation

template <**typename** VoxelType>
class **LayerTest**
Holds functions for comparing layers, blocks, voxels, etc for testing.

Public Functions

void **CompareLayers**(**const** *Layer*<VoxelType> &*layer_A*, **const** *Layer*<VoxelType> &*layer_B*) **const**

void **CompareBlocks**(**const** *Block*<VoxelType> &*block_A*, **const** *Block*<VoxelType> &*block_B*) **const**

void **CompareVoxel**(**const** VoxelType &*voxel_A*, **const** VoxelType &*voxel_B*) **const**

template <>
void **CompareVoxel**(**const** *EsdfVoxel* &*voxel_A*, **const** *EsdfVoxel* &*voxel_B*) **const**

template <>
void **CompareVoxel**(**const** *OccupancyVoxel* &*voxel_A*, **const** *OccupancyVoxel* &*voxel_B*) **const**

template <>
void **CompareVoxel**(**const** *TsdfVoxel* &*voxel_A*, **const** *TsdfVoxel* &*voxel_B*) **const**

template <>
void **CompareVoxel**(**const** *IntensityVoxel* &*voxel_A*, **const** *IntensityVoxel* &*voxel_B*) **const**

Public Static Attributes

constexpr double **kTolerance** = 1e-10

Class ThreadSafeIndex

- Defined in *File integrator_utils.h*

Class Documentation

class ThreadSafeIndex

Small class that can be used by multiple threads that need mutually exclusive indexes to the same array, while still covering all elements.

The class attempts to ensure that the points are read in an order that gives good coverage over the pointcloud very quickly. This is so that the integrator can be terminated before all points have been read (due to time constraints) and still capture most of the geometry.

Public Functions

ThreadSafeIndex(size_t *number_of_points*)

bool **getNextIndex**(size_t **idx*)

returns true if index is valid, false otherwise

void **reset**()

Template Class Accumulator

- Defined in *File timing.h*

Class Documentation

template <typename *T*, typename *Total*, int *N*>
class Accumulator

Public Functions

Accumulator()

void **Add**(T *sample*)

int **TotalSamples**() **const**

double **Sum**() **const**

double **Mean**() **const**

double **RollingMean**() **const**

double **Max**() **const**

double **Min**() **const**

double **LazyVariance**() **const**

Class DummyTimer

- Defined in *File timing.h*

Class Documentation

class DummyTimer

A class that has the timer interface but does nothing.

Swapping this in in place of the *Timer* class (say with a typedef) should allow one to disable timing. Because all of the functions are inline, they should just disappear.

Public Functions

DummyTimer(size_t, bool = false)

DummyTimer(std::string **const**&, bool = false)

~DummyTimer()

void **Start**()

void **Stop**()

bool **IsTiming**()

Class Timer

- Defined in *File timing.h*

Class Documentation

class Timer

Public Functions

Timer(size_t *handle*, bool *constructStopped* = false)

Timer(std::string **const** &*tag*, bool *constructStopped* = false)

~Timer()

void **Start**()

void **Stop**()

bool **IsTiming**() **const**

Class Timing

- Defined in *File timing.h*

Class Documentation

class Timing

Public Types

typedef std::map<std::string, size_t> **map_t**

Public Static Functions

static size_t **GetHandle**(std::string **const** &*tag*)

static std::string **GetTag**(size_t *handle*)

static double **GetTotalSeconds**(size_t *handle*)

static double **GetTotalSeconds**(std::string **const** &*tag*)

static double **GetMeanSeconds**(size_t *handle*)

static double **GetMeanSeconds**(std::string **const** &*tag*)

static size_t **GetNumSamples**(size_t *handle*)

static size_t **GetNumSamples**(std::string **const** &*tag*)

static double **GetVarianceSeconds**(size_t *handle*)

static double **GetVarianceSeconds**(std::string **const** &*tag*)

static double **GetMinSeconds**(size_t *handle*)

static double **GetMinSeconds**(std::string **const** &*tag*)

static double **GetMaxSeconds**(size_t *handle*)

static double **GetMaxSeconds**(std::string **const** &*tag*)

static double **GetHz**(size_t *handle*)

static double **GetHz**(std::string **const** &*tag*)

static void **Print**(std::ostream &*out*)

static std::string **Print**()

static std::string **SecondsToTimeString**(double *seconds*)

static void **Reset**()

static const *map_t* &**GetTimers**()

Friends

friend voxblox::timing::Timing::Timer

Class Transformer

- Defined in *File transformer.h*

Class Documentation

class Transformer

Class that binds to either the TF tree or resolves transformations from the ROS parameter server, depending on settings loaded from ROS params.

Public Functions

Transformer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*)

bool **lookupTransform**(**const** std::string &*from_frame*, **const** std::string &*to_frame*, **const** ros::Time &*timestamp*, *Transformation* **transform*)

void **transformCallback**(**const** geometry_msgs::TransformStamped &*transform_msg*)

Class TsdfIntegratorBase

- Defined in *File tsdf_integrator.h*

Nested Relationships

Nested Types

- *Struct TsdfIntegratorBase::Config*

Inheritance Relationships

Derived Types

- `public voxblox::FastTsdfIntegrator` (*Class FastTsdfIntegrator*)
- `public voxblox::MergedTsdfIntegrator` (*Class MergedTsdfIntegrator*)
- `public voxblox::SimpleTsdfIntegrator` (*Class SimpleTsdfIntegrator*)

Class Documentation

class TsdfIntegratorBase

Base class to the simple, merged and fast TSDF integrators.

The integrator takes in a pointcloud + pose and uses this information to update the TSDF information in the given TSDF layer. Note most functions in this class state if they are thread safe. Unless explicitly stated otherwise, this thread safety is based on the assumption that any pointers passed to the functions point to objects that are guaranteed to not be accessed by other threads.

Subclassed by *voxblox::FastTsdfIntegrator*, *voxblox::MergedTsdfIntegrator*, *voxblox::SimpleTsdfIntegrator*

Public Types

typedef std::shared_ptr<*TsdfIntegratorBase*> **Ptr**

Public Functions

TsdfIntegratorBase(**const** *Config* &*config*, *Layer*<*TsdfVoxel*> **layer*)

virtual void **integratePointCloud**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*points_C*, **const** *Colors* &*colors*, **const** bool *freespace_points* = false) = 0

Integrates the given point infomation into the TSDF.

NOT thread safe.

Parameters

- `freespace_points`: if true points will only be integrated up to the truncation distance. Used when we are given a minimum distance to a point, rather then exact distance. This is useful for clearing out free space.

const *Config* &**getConfig**() **const**

Returns a CONST ref of the config.

Protected Functions

bool **isPointValid**(**const** *Point* &*point_C*, **const** bool *freespace_point*, bool **is_clearing*) **const**

Thread safe.

TsdfVoxel ***allocateStorageAndGetVoxelPtr**(**const** *GlobalIndex* &*global_voxel_idx*, *Block*<*TsdfVoxel*>::Ptr **last_block*, *BlockIndex* **last_block_idx*)

Will return a pointer to a voxel located at global_voxel_idx in the tsdf layer.

Thread safe. Takes in the last_block_idx and last_block to prevent unneeded map lookups. If this voxel belongs to a block that has not been allocated, a block in temp_block_map_ is created/accessed and a voxel from this map is returned instead. Unlike the layer, accessing temp_block_map_ is controlled via a mutex allowing it to grow during integration. These temporary blocks can be merged into the layer later by calling updateLayerWithStoredBlocks

void **updateLayerWithStoredBlocks**()

Merges temporarily stored blocks into the main layer.

NOT thread safe, see allocateStorageAndGetVoxelPtr for more details.

void **updateTsdfVoxel**(**const** *Point* &*origin*, **const** *Point* &*point_G*, **const** *GlobalIndex* &*global_voxel_index*, **const** *Color* &*color*, **const** float *weight*, *TsdfVoxel* **tsdf_voxel*)

Updates tsdf_voxel, Thread safe.

float **computeDistance**(**const** *Point* &*origin*, **const** *Point* &*point_G*, **const** *Point* &*voxel_center*) **const**

Calculates TSDF distance, Thread safe.

float **getVoxelWeight**(**const** *Point* &*point_C*) **const**

Thread safe.

Protected Attributes

Config **config_**

Layer<*TsdfVoxel*> ***layer_**

FloatingPoint **voxel_size_**

size_t **voxels_per_side_**

FloatingPoint **block_size_**

FloatingPoint **voxel_size_inv_**

FloatingPoint **voxels_per_side_inv_**

FloatingPoint **block_size_inv_**

std::mutex **temp_block_mutex_**

Layer<*TsdfVoxel*>::BlockHashMap **temp_block_map_**
Temporary block storage, used to hold blocks that need to be created while integrating a new pointcloud.

ApproxHashArray<12, std::mutex, *GlobalIndex*, *LongIndexHash*> **mutexes_**
We need to prevent simultaneous access to the voxels in the map.

We could put a single mutex on the map or on the blocks, but as voxel updating is the most expensive operation in integration and most voxels are close together, both strategies would bottleneck the system. We could make a mutex per voxel, but this is too ram heavy as one mutex = 40 bytes. Because of this we create an array that is indexed by the first n bits of the voxels hash. Assuming a uniform hash distribution, this means the chance of two threads needing the same lock for unrelated voxels is (num_threads / (2^n)). For 8 threads and 12 bits this gives 0.2%.

struct Config

Public Members

float **default_truncation_distance** = 0.1

float **max_weight** = 10000.0

bool **voxel_carving_enabled** = true

FloatingPoint **min_ray_length_m** = 0.1

FloatingPoint **max_ray_length_m** = 5.0

bool **use_const_weight** = false

bool **allow_clear** = true

bool **use_weight_dropoff** = true

bool **use_sparsity_compensation_factor** = false

float **sparsity_compensation_factor** = 1.0f

size_t **integrator_threads** = std::thread::hardware_concurrency()

bool **enable_anti_grazing** = false
merge integrator specific

float **start_voxel_subsampling_factor** = 2.0f
fast integrator specific

int **max_consecutive_ray_collisions** = 2
fast integrator specific

int **clear_checks_every_n_frames** = 1
fast integrator specific

float **max_integration_time_s** = std::numeric_limits<float>::max()
fast integrator specific

Class TsdfIntegratorFactory

- Defined in *File tsdf_integrator.h*

Class Documentation

class TsdfIntegratorFactory
Creates a TSDF integrator of the desired type.

Public Static Functions

static *TsdfIntegratorBase*::*Ptr* **create**(**const** std::string &*integrator_type_name*, **const** *TsdfIntegratorBase*::*Config* &*config*, *Layer*<*TsdfVoxel*> **layer*)

static *TsdfIntegratorBase*::*Ptr* **create**(**const** *TsdfIntegratorType* *integrator_type*, **const** *TsdfIntegratorBase*::*Config* &*config*, *Layer*<*TsdfVoxel*> **layer*)

Class TsdfMap

- Defined in *File tsdf_map.h*

Nested Relationships

Nested Types

- *Struct TsdfMap::Config*

Class Documentation

class TsdfMap
Map holding a Truncated Signed Distance Field *Layer*.

Contains functions for interacting with the layer and getting gradient and distance information.

Public Types

typedef std::shared_ptr<*TsdfMap*> **Ptr**

using EigenDStride = Eigen::Stride<Eigen::Dynamic, Eigen::Dynamic>

using EigenDRef = Eigen::Ref<MatrixType, 0, *EigenDStride*>

Public Functions

TsdfMap(**const** *Config* &*config*)

TsdfMap(**const** *Layer*<*TsdfVoxel*> &*layer*)
Creates a new *TsdfMap* based on a COPY of this layer.

TsdfMap(*Layer*<*TsdfVoxel*>::Ptr *layer*)
Creates a new *TsdfMap* that contains this layer.

virtual ~TsdfMap()

Layer<*TsdfVoxel*> ***getTsdfLayerPtr**()

const *Layer*<*TsdfVoxel*> &**getTsdfLayer**() **const**

FloatingPoint **block_size**() **const**

FloatingPoint **voxel_size**() **const**

unsigned int **coordPlaneSliceGetDistanceWeight**(unsigned int *free_plane_index*, double *free_plane_val*, *EigenDRef*<Eigen::Matrix<double, 3, Eigen::Dynamic>> &*positions*, Eigen::Ref<Eigen::VectorXd> *distances*, Eigen::Ref<Eigen::VectorXd> *weights*, unsigned int *max_points*) **const**
Extract all voxels on a slice plane that is parallel to one of the axis-aligned planes.

free_plane_index specifies the free coordinate (zero-based; x, y, z order) free_plane_val specifies the plane intercept coordinate along that axis

Protected Attributes

FloatingPoint **block_size_**

Layer<*TsdfVoxel*>::Ptr **tsdf_layer_**

struct Config

Public Members

FloatingPoint **tsdf_voxel_size** = 0.2

size_t **tsdf_voxels_per_side** = 16u

Class TsdfServer

- Defined in *File tsdf_server.h*

Inheritance Relationships

Derived Types

- `public voxblox::EsdfServer` (*Class EsdfServer*)
- `public voxblox::IntensityServer` (*Class IntensityServer*)

Class Documentation

class TsdfServer

Subclassed by *voxblox::EsdfServer*, *voxblox::IntensityServer*

Public Functions

TsdfServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*)

TsdfServer(**const** ros::NodeHandle &*nh*, **const** ros::NodeHandle &*nh_private*, **const** *TsdfMap*::*Config* &*config*, **const** *TsdfIntegratorBase*::*Config* &*integrator_config*)

virtual ~TsdfServer()

void **getServerConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

void **insertPointcloud**(**const** sensor_msgs::PointCloud2::Ptr &*pointcloud*)

void **insertFreespacePointcloud**(**const** sensor_msgs::PointCloud2::Ptr &*pointcloud*)

virtual void **processPointCloudMessageAndInsert**(**const** sensor_msgs::PointCloud2::Ptr &*pointcloud_msg*, **const** *Transformation* &*T_G_C*, **const** bool *is_freespace_pointcloud*)

void **integratePointcloud**(**const** *Transformation* &*T_G_C*, **const** *Pointcloud* &*ptcloud_C*, **const** *Colors* &*colors*, **const** bool *is_freespace_pointcloud* = false)

virtual void **newPoseCallback**(**const** *Transformation*&)

void **publishAllUpdatedTsdfVoxels**()

void **publishTsdfSurfacePoints**()

void **publishTsdfOccupiedNodes**()

virtual void **publishSlices**()

virtual void **updateMesh**()
Incremental update.

virtual bool **generateMesh**()
Batch update.

virtual void **publishPointclouds**()

virtual void **publishMap**(**const** bool *reset_remote_map* = false)

virtual bool **saveMap**(**const** std::string &*file_path*)

virtual bool **loadMap**(**const** std::string &*file_path*)

bool **clearMapCallback**(std_srvs::Empty::Request &*request*, std_srvs::Empty::Response &*response*)

bool **saveMapCallback**(voxblox_msgs::FilePath::Request &*request*, voxblox_msgs::FilePath::Response &*response*)

bool **loadMapCallback**(voxblox_msgs::FilePath::Request &*request*, voxblox_msgs::FilePath::Response &*response*)

bool **generateMeshCallback**(std_srvs::Empty::Request &*request*, std_srvs::Empty::Response &*response*)

bool **publishPointcloudsCallback**(std_srvs::Empty::Request &*request*, std_srvs::Empty::Response &*response*)

bool **publishTsdfMapCallback**(std_srvs::Empty::Request &*request*, std_srvs::Empty::Response &*response*)

void **updateMeshEvent**(**const** ros::TimerEvent &*event*)

std::shared_ptr<*TsdfMap*> **getTsdfMapPtr**()

double **getSliceLevel**() **const**
Accessors for setting and getting parameters.

void **setSliceLevel**(double *slice_level*)

bool **setPublishSlices**() **const**

void **setPublishSlices**(**const** bool *publish_slices*)

void **setWorldFrame**(**const** std::string &*world_frame*)

std::string **getWorldFrame**() **const**

virtual void **clear**()
CLEARS THE ENTIRE MAP!

void **tsdfMapCallback**(**const** voxblox_msgs::Layer &*layer_msg*)
Overwrites the layer with what's coming from the topic!

Protected Functions

bool **getNextPointcloudFromQueue**(std::queue<sensor_msgs::PointCloud2::Ptr> **queue*, sensor_msgs::PointCloud2::Ptr **pointcloud_msg*, *Transformation* **T_G_C*)
Gets the next pointcloud that has an available transform to process from the queue.

Protected Attributes

ros::NodeHandle **nh_**

ros::NodeHandle **nh_private_**

bool **verbose_**

std::string **world_frame_**
Global/map coordinate frame.

Will always look up TF transforms to this frame.

std::string **icp_corrected_frame_**
Name of the *ICP* corrected frame.

Publishes TF and transform topic to this if *ICP* on.

std::string **pose_corrected_frame_**
Name of the pose in the *ICP* correct Frame.

double **max_block_distance_from_body_**
Delete blocks that are far from the system to help manage memory.

double **slice_level_**
Pointcloud visualization settings.

bool **use_freespace_pointcloud_**
If the system should subscribe to a pointcloud giving points in freespace.

std::string **mesh_filename_**
Mesh output settings.

Mesh is only written to file if mesh_filename_ is not empty.

ColorMode **color_mode_**
How to color the mesh.

std::unique_ptr<*ColorMap*> **color_map_**
Colormap to use for intensity pointclouds.

ros::Duration **min_time_between_msgs_**
Will throttle to this message rate.

bool **publish_tsdf_info_**
What output information to publish.

bool **publish_slices_**

bool **publish_pointclouds_**

bool **publish_tsdf_map_**

bool **cache_mesh_**
Whether to save the latest mesh message sent (for inheriting classes).

bool **enable_icp_**
Whether to enable *ICP* corrections.

Every pointcloud coming in will attempt to be matched up to the existing structure using *ICP*. Requires the initial guess from odometry to already be very good.

bool **accumulate_icp_corrections_**
If using *ICP* corrections, whether to store accumulate the corrected transform.

If this is set to false, the transform will reset every iteration.

ros::Subscriber **pointcloud_sub_**
Data subscribers.

ros::Subscriber **freespace_pointcloud_sub_**

int **pointcloud_queue_size_**
Subscriber settings.

ros::Publisher **mesh_pub_**

ros::Publisher **tsdf_pointcloud_pub_**

ros::Publisher **surface_pointcloud_pub_**

ros::Publisher **tsdf_slice_pub_**

ros::Publisher **occupancy_marker_pub_**

ros::Publisher **icp_transform_pub_**

ros::Publisher **tsdf_map_pub_**
Publish the complete map for other nodes to consume.

ros::Subscriber **tsdf_map_sub_**
Subscriber to subscribe to another node generating the map.

ros::ServiceServer **generate_mesh_srv_**

ros::ServiceServer **clear_map_srv_**

ros::ServiceServer **save_map_srv_**

ros::ServiceServer **load_map_srv_**

ros::ServiceServer **publish_pointclouds_srv_**

ros::ServiceServer **publish_tsdf_map_srv_**

tf::TransformBroadcaster **tf_broadcaster_**
Tools for broadcasting TFs.

ros::Timer **update_mesh_timer_**

std::shared_ptr<*TsdfMap*> **tsdf_map_**

std::unique_ptr<*TsdfIntegratorBase*> **tsdf_integrator_**

std::shared_ptr<*ICP*> **icp_**
ICP matcher.

std::shared_ptr<*MeshLayer*> **mesh_layer_**

std::unique_ptr<*MeshIntegrator*<*TsdfVoxel*>> **mesh_integrator_**

voxblox_msgs::Mesh **cached_mesh_msg_**
Optionally cached mesh message.

Transformer **transformer_**
Transformer object to keep track of either TF transforms or messages from a transform topic.

std::queue<sensor_msgs::PointCloud2::Ptr> **pointcloud_queue_**
Queue of incoming pointclouds, in case the transforms can't be immediately resolved.

std::queue<sensor_msgs::PointCloud2::Ptr> **freespace_pointcloud_queue_**

ros::Time **last_msg_time_ptcloud_**

ros::Time **last_msg_time_freespace_ptcloud_**

Transformation **icp_corrected_transform_**
Current transform corrections from *ICP*.

Class VoxbloxMeshDisplay

- Defined in *File voxblox_mesh_display.h*

Inheritance Relationships

Base Type

- `public rviz::MessageFilterDisplay< voxblox_msgs::Mesh >`

Class Documentation

class VoxbloxMeshDisplay : **public** rviz::MessageFilterDisplay<voxblox_msgs::Mesh>

Public Functions

VoxbloxMeshDisplay()

virtual ~VoxbloxMeshDisplay()

Protected Functions

virtual void **onInitialize**()

virtual void **reset**()

Class VoxbloxMeshVisual

- Defined in *File voxblox_mesh_visual.h*

Class Documentation

class VoxbloxMeshVisual

Visualizes a single voxblox_msgs::Mesh message.

Public Functions

VoxbloxMeshVisual(Ogre::SceneManager **scene_manager*, Ogre::SceneNode **parent_node*)

virtual ~VoxbloxMeshVisual()

void **setMessage**(**const** voxblox_msgs::Mesh::ConstPtr &*msg*)

void **setFramePosition**(**const** Ogre::Vector3 &*position*)

Set the coordinate frame pose.

void **setFrameOrientation**(**const** Ogre::Quaternion &*orientation*)

10.3.3 Enums

Enum ColorMode

Enum Documentation

enum voxblox::**ColorMode**

Values:

kColor = 0

kHeight

kNormals

kGray

kLambert

kLambertColor

Enum Connectivity

- Defined in *File neighbor_tools.h*

Enum Documentation

enum voxblox::**Connectivity**

Define what connectivity you want to have in the voxels.

Values:

kSix = 6u

kEighteen = 18u

kTwentySix = 26u

Enum PlyOutputTypes

- Defined in *File sdf_ply.h*

Enum Documentation

enum voxblox::io::**PlyOutputTypes**

Values:

kSdfColoredDistanceField

kSdfIsosurface

kSdfIsosurfaceConnected

Enum MapDerializationAction

Enum Documentation

enum voxblox::**MapDerializationAction**
Values:

kUpdate = 0u

kMerge = 1u

kReset = 2u

Enum TsdfIntegratorType

- Defined in *File tsdf_integrator.h*

Enum Documentation

enum voxblox::**TsdfIntegratorType**
Values:

kSimple = 1

kMerged = 2

kFast = 3

Enum VoxelEvaluationMode

Enum Documentation

enum voxblox::utils::**VoxelEvaluationMode**
Values:

kEvaluateAllVoxels

kIgnoreErrorBehindTestSurface

kIgnoreErrorBehindGtSurface

kIgnoreErrorBehindAllSurfaces

Enum VoxelEvaluationResult

- Defined in *File evaluation_utils.h*

Enum Documentation

enum voxblox::utils::**VoxelEvaluationResult**
Values:

kNoOverlap

kIgnored

kEvaluated

10.3.4 Functions

Template Function voxblox::aligned_shared

Function Documentation

template <typename Type, typename... *Arguments*>
std::shared_ptr<Type> voxblox::**aligned_shared**(Arguments&&... *arguments*)

Function voxblox::castRay

Function Documentation

void voxblox::**castRay**(**const** *Point* &*start_scaled*, **const** *Point* &*end_scaled*, AlignedVector<*GlobalIndex*> **indices*)

This function assumes PRE-SCALED coordinates, where one unit = one voxel size.

The indices are also returned in this scales coordinate system, which should map to voxel indices.

Function voxblox::colorMsgToVoxblox

Function Documentation

void voxblox::**colorMsgToVoxblox**(**const** std_msgs::ColorRGBA &*color_msg*, *Color* **color*)

Function voxblox::colorVoxbloxToMsg

Function Documentation

void voxblox::**colorVoxbloxToMsg**(**const** *Color* &*color*, std_msgs::ColorRGBA **color_msg*)

Function voxblox::convertMeshLayerToMesh

Function Documentation

bool voxblox::convertMeshLayerToMesh(const MeshLayer & mesh_layer, Mesh * mesh, const bo

Generates a mesh from the mesh layer.

Parameters

- connected_mesh: if true veracities will be shared between triangles
- vertex_proximity_threshold: verticies that are within the specified thershold distance will be merged together, simplifying the mesh.

Template Function voxblox::createColorPointcloudFromLayer(const Layer<VoxelType>&, const ShouldVisualizeVoxelColorFunctionType<VoxelType>&, pcl::PointCloud<pcl::PointXYZRGB> *)

- Defined in *File ptcloud_vis.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**createColorPointcloudFromLayer**(**const** *Layer*<VoxelType> &*layer*, **const** ShouldVisualizeVoxelColorFunctionType<VoxelType> &*vis_function*, pcl::PointCloud<pcl::PointXYZRGB> **pointcloud*)

Template function to visualize a colored pointcloud.

Template Function voxblox::createColorPointcloudFromLayer(const Layer<VoxelType>&, const ShouldVisualizeVoxelIntensityFunctionType<VoxelType>&, pcl::PointCloud<pcl::PointXYZI> *)

Function Documentation

template <**typename** VoxelType>
void voxblox::**createColorPointcloudFromLayer**(**const** *Layer*<VoxelType> &*layer*, **const** ShouldVisualizeVoxelIntensityFunctionType<VoxelType> &*vis_function*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Template function to visualize an intensity pointcloud.

Function voxblox::createConnectedMesh(const AlignedVector<Mesh::ConstPtr>&, Mesh *, const FloatingPoint)

Function Documentation

void voxblox::createConnectedMesh(const AlignedVector< Mesh::ConstPtr > & meshes, Mesh

Combines all given meshes into a single mesh with connected verticies.

Also removes triangles with zero surface area. If you only would like to connect vertices, make sure that the proximity threhsold <<< voxel size. If you would like to simplify the mesh, chose a threshold greater or near the voxel size until you reached the level of simpliciation desired.

Function voxblox::createConnectedMesh(const Mesh&, Mesh *, const FloatingPoint)

Function Documentation

void voxblox::createConnectedMesh(const Mesh & mesh, Mesh * connected_mesh, const Float

Function voxblox::createDistancePointcloudFromEsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createDistancePointcloudFromEsdfLayer**(**const** *Layer*<*EsdfVoxel*> &*layer*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Function voxblox::createDistancePointcloudFromEsdfLayerSlice

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createDistancePointcloudFromEsdfLayerSlice**(**const** *Layer*<*EsdfVoxel*> &*layer*, unsigned int *free_plane_index*, *FloatingPoint* *free_plane_val*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Function voxblox::createDistancePointcloudFromTsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createDistancePointcloudFromTsdfLayer**(**const** *Layer*<*TsdfVoxel*> &*layer*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Create a pointcloud based on all the TSDF voxels.

The intensity is determined based on the distance to the surface.

Function voxblox::createDistancePointcloudFromTsdfLayerSlice

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createDistancePointcloudFromTsdfLayerSlice**(**const** *Layer*<*TsdfVoxel*> &*layer*, unsigned int *free_plane_index*, *FloatingPoint* *free_plane_val*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Function voxblox::createFreePointcloudFromEsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createFreePointcloudFromEsdfLayer**(**const** *Layer*<*EsdfVoxel*> &*layer*, float *min_distance*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Function voxblox::createIntensityPointcloudFromIntensityLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createIntensityPointcloudFromIntensityLayer**(**const** *Layer*<*IntensityVoxel*> &*layer*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Template Function voxblox::createOccupancyBlocksFromLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**createOccupancyBlocksFromLayer**(**const** *Layer*<VoxelType> &*layer*, **const** ShouldVisualizeVoxelFunctionType<VoxelType> &*vis_function*, **const** std::string &*frame_id*, visualization_msgs::MarkerArray **marker_array*)

Function voxblox::createOccupancyBlocksFromOccupancyLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createOccupancyBlocksFromOccupancyLayer**(**const** *Layer*<*OccupancyVoxel*> &*layer*, **const** std::string &*frame_id*, visualization_msgs::MarkerArray **marker_array*)

Function voxblox::createOccupancyBlocksFromTsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createOccupancyBlocksFromTsdfLayer**(**const** *Layer*<*TsdfVoxel*> &*layer*, **const** std::string &*frame_id*, visualization_msgs::MarkerArray **marker_array*)

Function voxblox::createPointcloudFromTsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createPointcloudFromTsdfLayer**(**const** *Layer*<*TsdfVoxel*> &*layer*, pcl::PointCloud<pcl::PointXYZRGB> **pointcloud*)

Create a pointcloud based on all the TSDF voxels.

The RGB color is determined by the color of the TSDF voxel.

Function voxblox::createSurfaceDistancePointcloudFromTsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createSurfaceDistancePointcloudFromTsdfLayer**(**const** *Layer*<*TsdfVoxel*> &*layer*, double *surface_distance*, pcl::PointCloud<pcl::PointXYZI> **pointcloud*)

Create a pointcloud based on the TSDF voxels near the surface.

The intensity is determined based on the distance to the surface.

Function voxblox::createSurfacePointcloudFromTsdfLayer

- Defined in *File ptcloud_vis.h*

Function Documentation

void voxblox::**createSurfacePointcloudFromTsdfLayer**(**const** *Layer*<*TsdfVoxel*> &*layer*, double *surface_distance*, pcl::PointCloud<pcl::PointXYZRGB> **pointcloud*)

Create a pointcloud based on the TSDF voxels near the surface.

The RGB color is determined by the color of the TSDF voxel.

Template Function voxblox::deserializeMsgToLayer(const voxblox_msgs::Layer&, Layer<VoxelType> *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::**deserializeMsgToLayer**(**const** voxblox_msgs::Layer &*msg*, *Layer*<VoxelType> **layer*)

Returns true if could parse the data into the existing layer (all parameters are compatible), false otherwise.

This function will use the deserialization action suggested by the layer message.

Template Function voxblox::deserializeMsgToLayer(const voxblox_msgs::Layer&, const MapDerializationAction&, Layer<VoxelType> *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::**deserializeMsgToLayer**(**const** voxblox_msgs::Layer &*msg*, **const** *MapDerializationAction* &*action*, *Layer*<VoxelType> **layer*)

Template Function voxblox::evaluateLayerRmseAtPoses(const utils::VoxelEvaluationMode&, const Layer<VoxelType>&, const Layer<VoxelType>&, const std::vector<Transformation>&, std::vector<utils::VoxelEvaluationDetails> *, std::vector<std::pair<typename voxblox::Layer<VoxelType>::Ptr, typename voxblox::Layer<VoxelType>::Ptr>> *)

- Defined in *File merge_integration.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**evaluateLayerRmseAtPoses**(**const** utils::*VoxelEvaluationMode* &*voxel_evaluation_mode*, **const** *Layer*<VoxelType> &*layer_A*, **const** *Layer*<VoxelType> &*layer_B*, **const** std::vector<*Transformation*> &*transforms_A_B*, std::vector<utils::*VoxelEvaluationDetails*> **voxel_evaluation_details_vector*, std::vector<std::pair<**typename** voxblox::*Layer*<VoxelType>::Ptr, **typename** voxblox::*Layer*<VoxelType>::Ptr>> **aligned_layers_and_error_layers* = nullptr)

This function will align layer B to layer A, transforming and interpolating layer B into voxel grid A for every transformation in `transforms_A_B` and evaluate them.

If `aligned_layers_and_error_layers` is set, this function returns a vector containing the aligned layer_B and an error layer for every transformation. The error layer contains the absolute SDF error for every voxel of the comparison between layer_A and aligned layer_B. This function currently only supports SDF type layers, like *TsdfVoxel* and *EsdfVoxel*.

Template Function voxblox::evaluateLayerRmseAtPoses(const utils::VoxelEvaluationMode&, const Layer<VoxelType>&, const Layer<VoxelType>&, const std::vector<Eigen::Matrix<float, 4, 4>, Eigen::aligned_allocator<Eigen::Matrix<float, 4, 4>>>&, std::vector<utils::VoxelEvaluationDetails> *, std::vector<std::pair<typename voxblox::Layer<VoxelType>::Ptr, typename voxblox::Layer<VoxelType>::Ptr>> *)

Function Documentation

template <**typename** VoxelType>
void voxblox::**evaluateLayerRmseAtPoses**(**const** utils::*VoxelEvaluationMode* &*voxel_evaluation_mode*, **const** *Layer*<VoxelType> &*layer_A*, **const** *Layer*<VoxelType> &*layer_B*, **const** std::vector<Eigen::Matrix<float, 4, 4>, Eigen::aligned_allocator<Eigen::Matrix<float, 4, 4>>> &*transforms_A_B*, std::vector<utils::*VoxelEvaluationDetails*> **voxel_evaluation_details_vector*, std::vector<std::pair<**typename** voxblox::*Layer*<VoxelType>::Ptr, **typename** voxblox::*Layer*<VoxelType>::Ptr>> **aligned_layers_and_error_layers* = nullptr)

Function voxblox::fillMarkerWithMesh

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**fillMarkerWithMesh**(**const** *MeshLayer*::*ConstPtr* &*mesh_layer*, *ColorMode* *color_mode*, visualization_msgs::Marker **marker*)

Function voxblox::fillPointcloudWithMesh

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**fillPointcloudWithMesh**(**const** *MeshLayer*::*ConstPtr* &*mesh_layer*, *ColorMode* *color_mode*, pcl::PointCloud<pcl::PointXYZRGB> **pointcloud*)

Function voxblox::generateVoxbloxMeshMsg

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**generateVoxbloxMeshMsg**(**const** *MeshLayer*::*Ptr* &*mesh_layer*, *ColorMode* *color_mode*, voxblox_msgs::Mesh **mesh_msg*)

Function voxblox::getBlockAndVoxelIndexFromGlobalVoxelIndex

Function Documentation

void voxblox::**getBlockAndVoxelIndexFromGlobalVoxelIndex**(**const** *GlobalIndex* &*global_voxel_idx*, **const** int *voxels_per_side*, *BlockIndex* **block_index*, *VoxelIndex* **voxel_index*)

Function voxblox::getBlockIndexFromGlobalVoxelIndex

Function Documentation

BlockIndex voxblox::**getBlockIndexFromGlobalVoxelIndex**(**const** *GlobalIndex* &*global_voxel_idx*, *FloatingPoint* *voxels_per_side_inv*)

Template Function voxblox::getCenterPointFromGridIndex

Function Documentation

template <**typename** IndexType>
Point voxblox::**getCenterPointFromGridIndex**(**const** IndexType &*idx*, *FloatingPoint* *grid_size*)

Function voxblox::getEsdfIntegratorConfigFromRosParam

- Defined in *File ros_params.h*

Function Documentation

EsdfIntegrator::*Config* voxblox::**getEsdfIntegratorConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

Function voxblox::getEsdfMapConfigFromRosParam

- Defined in *File ros_params.h*

Function Documentation

EsdfMap::*Config* voxblox::**getEsdfMapConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

Function voxblox::getGlobalVoxelIndexFromBlockAndVoxelIndex

- Defined in *File common.h*

Function Documentation

GlobalIndex voxblox::**getGlobalVoxelIndexFromBlockAndVoxelIndex**(**const** *BlockIndex* &*block_index*, **const** *VoxelIndex* &*voxel_index*, int *voxels_per_side*)

Converts between *Block* + Voxel index and GlobalVoxelIndex.

Note that this takes int VOXELS_PER_SIDE, and getBlockIndexFromGlobalVoxelIndex takes voxels per side inverse.

Template Function voxblox::getGridIndexFromOriginPoint

- Defined in *File common.h*

Function Documentation

template <**typename** IndexType>
IndexType voxblox::**getGridIndexFromOriginPoint**(**const** *Point* &*point*, **const** *FloatingPoint* *grid_size_inv*)

NOTE: This function is safer than `getGridIndexFromPoint`, because it assumes we pass in not an arbitrary point in the grid cell, but the ORIGIN.

This way we can avoid the floating point precision issue that arrises for calls to `getGridIndexFromPoint` for arbitrary points near the border of the grid cell.

Template Function voxblox::getGridIndexFromPoint(const Point&, const FloatingPoint)

- Defined in *File common.h*

Function Documentation

template <**typename** IndexType>
IndexType voxblox::**getGridIndexFromPoint**(**const** *Point* &*point*, **const** *FloatingPoint* *grid_size_inv*)

NOTE: Due the limited accuracy of the FloatingPoint type, this function doesn't always compute the correct grid index for coordinates near the grid cell boundaries.

Use the safer `getGridIndexFromOriginPoint` if the origin point is available.

Template Function voxblox::getGridIndexFromPoint(const Point&)

Function Documentation

template <**typename** IndexType>
IndexType voxblox::**getGridIndexFromPoint**(**const** *Point* &*scaled_point*)

NOTE: Due the limited accuracy of the FloatingPoint type, this function doesn't always compute the correct grid index for coordinates near the grid cell boundaries.

Function voxblox::getHierarchicalIndexAlongRay

- Defined in *File integrator_utils.h*

Function Documentation

void voxblox::**getHierarchicalIndexAlongRay**(**const** *Point* &*start*, **const** *Point* &*end*, size_t *voxels_per_side*, *FloatingPoint* *voxel_size*, *FloatingPoint* *truncation_distance*, bool *voxel_carving_enabled*, *HierarchicalIndexMap* **hierarchical_idx_map*)

Takes start and end in WORLD COORDINATES, does all pre-scaling and sorting into hierarhical index.

Function voxblox::getICPConfigFromRosParam

- Defined in *File ros_params.h*

Function Documentation

ICP::*Config* voxblox::**getICPConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

Function voxblox::getLocalFromGlobalVoxelIndex

Function Documentation

VoxelIndex voxblox::**getLocalFromGlobalVoxelIndex**(**const** *GlobalIndex* &*global_voxel_idx*, **const** int *voxels_per_side*)

Converts from a global voxel index to the index inside a block.

NOTE: assumes that voxels_per_side is a power of 2 and uses a bitwise and as a computationally cheap substitute for the modulus operator

Template Function voxblox::getOriginPointFromGridIndex

Function Documentation

template <**typename** IndexType>
Point voxblox::**getOriginPointFromGridIndex**(**const** IndexType &*idx*, *FloatingPoint* *grid_size*)

Template Function voxblox::getSurfaceDistanceAlongRay

- Defined in *File distance_utils.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::**getSurfaceDistanceAlongRay**(**const** *Layer*<VoxelType> &*layer*, **const** *Point* &*ray_origin*, **const** *Point* &*bearing_vector*, *FloatingPoint* *max_distance*, *Point* **triangulated_pose*)

Returns true if there is a valid distance (intersection with the surface), false otherwise (no known space, no surface boundary, etc.).

VoxelType must have a distance defined.

Function voxblox::getTsdfIntegratorConfigFromRosParam

- Defined in *File ros_params.h*

Function Documentation

TsdfIntegratorBase::*Config* voxblox::**getTsdfIntegratorConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

Function voxblox::getTsdfMapConfigFromRosParam

- Defined in *File ros_params.h*

Function Documentation

TsdfMap::*Config* voxblox::**getTsdfMapConfigFromRosParam**(**const** ros::NodeHandle &*nh_private*)

Function voxblox::getVertexColor

- Defined in *File mesh_vis.h*

Function Documentation

std_msgs::ColorRGBA voxblox::**getVertexColor**(**const** *Mesh*::*ConstPtr* &*mesh*, **const** *ColorMode* &*color_mode*, **const** size_t *index*)

Template Function voxblox::getVoxelType

- Defined in *File voxel.h*

Function Documentation

template <**typename** Type>
std::string voxblox::**getVoxelType**()

Function voxblox::getVoxelType< EsdfVoxel >

- Defined in *File voxel.h*

Function Documentation

template <>
template<>
std::string voxblox::**getVoxelType**<*EsdfVoxel*>()

Function voxblox::getVoxelType< IntensityVoxel >

- Defined in *File voxel.h*

Function Documentation

template <>
template<>
std::string voxblox::**getVoxelType**<*IntensityVoxel*>()

Function voxblox::getVoxelType< OccupancyVoxel >

- Defined in *File voxel.h*

Function Documentation

template <>
template<>
std::string voxblox::**getVoxelType**<*OccupancyVoxel*>()

Function voxblox::getVoxelType< TsdfVoxel >

- Defined in *File voxel.h*

Function Documentation

template <>
template<>
std::string voxblox::**getVoxelType**<*TsdfVoxel*>()

Function voxblox::grayColorMap

Function Documentation

Color voxblox::**grayColorMap**(double *h*)
Maps an input h from a value between 0.0 and 1.0 into a grayscale color.

Function voxblox::heightColorFromVertex

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**heightColorFromVertex**(**const** *Point* &*vertex*, std_msgs::ColorRGBA **color_msg*)

Template Function voxblox::io::convertLayerToMesh(const Layer<VoxelType>&, const MeshIntegratorConfig&, voxblox::Mesh *, const bool, const FloatingPoint)

- Defined in *File sdf_ply.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::convertLayerToMesh(const Layer < VoxelType > & layer, const MeshIntegra

Converts the layer to a mesh by extracting its ISO surface using marching cubes.

This function returns false if the mesh is empty. The mesh can either be extracted as a set of distinct triangles, or the function can try to connect all identical vertices to create a connected mesh.

Template Function voxblox::io::convertLayerToMesh(const Layer<VoxelType>&, voxblox::Mesh *, const bool, const FloatingPoint)

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::convertLayerToMesh(const Layer < VoxelType > & layer, voxblox::Mesh * r

Template Function voxblox::io::convertVoxelGridToPointCloud(const Layer<VoxelType>&, const float, const float, voxblox::Mesh *)

- Defined in *File sdf_ply.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::**io::convertVoxelGridToPointCloud**(**const** *Layer*<VoxelType> &*layer*, **const** float *sdf_color_range*, **const** float *sdf_max_value*, voxblox::*Mesh* **point_cloud*)

This function converts all voxels with positive weight/observed into points colored by a color map based on the SDF value.

The parameter sdf_color_range is used to determine the range of the rainbow color map which is used to visualize the SDF values. If an SDF value is outside this range, it will be truncated to the limits of the range. sdf_max_value determines if a point is generated for this value or not. Only SDF values within this max value

result in a colored point. If this threshold is set to a negative value, all points will be generated independent of the SDF value.

Template Function voxblox::io::convertVoxelGridToPointCloud(const Layer<VoxelType>&, const float, voxblox::Mesh *)

Function Documentation

```
template <typename VoxelType>
bool voxblox::io::convertVoxelGridToPointCloud(const Layer<VoxelType> &layer,
                                               const float sdf_color_range,
                                               voxblox::Mesh *point_cloud)
```

Template Function voxblox::io::getColorFromVoxel(const VoxelType&, const float, const float, Color *)

- Defined in *File sdf_ply.h*

Function Documentation

```
template <typename VoxelType>
bool voxblox::io::getColorFromVoxel(const VoxelType &voxel, const float sdf_color_range,
                                    const float sdf_max_value, Color *color)
```

Convert a voxel to a colored point.

The sdf_color_range determines the range that is covered by the rainbow colors. All absolute distance values that exceed this range will receive tha max/min color of the rainbow range. The sdf_max_value determines if a point is generated for this value or not. Only SDF values within this max value result in a colored point.

Function voxblox::io::getColorFromVoxel(const TsdfVoxel&, const float, const float, Color *)

Function Documentation

```
template <>
bool voxblox::io::getColorFromVoxel(const TsdfVoxel &voxel, const float sdf_color_range,
                                    const float sdf_max_value, Color *color)
```

Convert a voxel to a colored point.

The sdf_color_range determines the range that is covered by the rainbow colors. All absolute distance values that exceed this range will receive tha max/min color of the rainbow range. The sdf_max_value determines if a point is generated for this value or not. Only SDF values within this max value result in a colored point.

Function voxblox::io::getColorFromVoxel(const EsdfVoxel&, const float, const float, Color *)

Function Documentation

```
template <>
bool voxblox::io::getColorFromVoxel(const EsdfVoxel &voxel, const float sdf_color_range,
                                    const float sdf_max_value, Color *color)
```

Convert a voxel to a colored point.

The sdf_color_range determines the range that is covered by the rainbow colors. All absolute distance values that exceed this range will receive tha max/min color of the rainbow range. The sdf_max_value determines if a point is generated for this value or not. Only SDF values within this max value result in a colored point.

Template Function voxblox::io::LoadBlocksFromFile(const std::string&, typename Layer<VoxelType>::BlockMergingStrategy, Layer<VoxelType> *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**LoadBlocksFromFile**(**const** std::string &*file_path*, **typename** *Layer*<VoxelType>::BlockMergingStrategy *strategy*, *Layer*<VoxelType> **layer_ptr*)

By default, loads blocks without multiple layer support.

Loading blocks assumes that the layer is already setup and allocated.

Template Function voxblox::io::LoadBlocksFromFile(const std::string&, typename Layer<VoxelType>::BlockMergingStrategy, bool, Layer<VoxelType> *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**LoadBlocksFromFile**(**const** std::string &*file_path*, **typename** *Layer*<VoxelType>::BlockMergingStrategy *strategy*, bool *multiple_layer_support*, *Layer*<VoxelType> **layer_ptr*)

By default, loads blocks without multiple layer support.

Loading the full layer allocates a layer of the correct size.

Template Function voxblox::io::LoadLayer(const std::string&, typename Layer<VoxelType>::Ptr *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**LoadLayer**(**const** std::string &*file_path*, **typename** *Layer*<VoxelType>::Ptr **layer_ptr*)

Unlike LoadBlocks, this actually allocates the layer as well.

By default loads without multiple layer support (i.e., only checks the first layer in the file).

Template Function voxblox::io::LoadLayer(const std::string&, const bool, typename Layer<VoxelType>::Ptr *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**LoadLayer**(**const** std::string &*file_path*, **const** bool *multiple_layer_support*, **typename** *Layer*<VoxelType>::Ptr **layer_ptr*)

Unlike LoadBlocks, this actually allocates the layer as well.

By default loads without multiple layer support (i.e., only checks the first layer in the file).

Template Function voxblox::io::outputLayerAsPly

- Defined in *File sdf_ply.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**outputLayerAsPly**(**const** *Layer*<VoxelType> &*layer*, **const** std::string &*file-name*, *PlyOutputTypes type*, **const** float *sdf_color_range* = 0.3f, **const** float *max_sdf_value_to_output* = 0.3f)

Output the layer to ply file.

Depending on the ply output type, this either exports all voxel centers colored by th SDF values or extracts the ISO surface as mesh. The parameter sdf_color_range is used to color the points for modes that use an SDF-based point cloud coloring function.

Template Function voxblox::io::SaveLayer

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**SaveLayer**(**const** *Layer*<VoxelType> &*layer*, **const** std::string &*file_path*, bool *clear_file* = true)

By default, clears (truncates) the output file.

Set clear_file to false in case writing the second (or subsequent) layer into the same file.

Template Function voxblox::io::SaveLayerSubset

Function Documentation

template <**typename** VoxelType>
bool voxblox::io::**SaveLayerSubset**(**const** *Layer*<VoxelType> &*layer*, **const** std::string &*file_path*, **const** *BlockIndexList* &*blocks_to_include*, bool *include_all_blocks*)

Saves only some parts of the layer to the file. Clears the file by default.

Function voxblox::isPowerOfTwo

Function Documentation

bool voxblox::**isPowerOfTwo**(int *x*)

Function voxblox::lambertColorFromColorAndNormal

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**lambertColorFromColorAndNormal**(**const** *Color* &*color*, **const** *Point* &*normal*, std_msgs::ColorRGBA **color_msg*)

Function voxblox::lambertColorFromNormal

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**lambertColorFromNormal**(**const** *Point* &*normal*, std_msgs::ColorRGBA **color_msg*)

Function voxblox::lambertShading

- Defined in *File mesh_vis.h*

Function Documentation

Point voxblox::**lambertShading**(**const** *Point* &*normal*, **const** *Point* &*light*, **const** *Point* &*color*)

Function voxblox::logOddsFromProbability

Function Documentation

float voxblox::**logOddsFromProbability**(float *probability*)

Template Function voxblox::mergeLayerAintoLayerB(const Layer<VoxelType>&, Layer<VoxelType> *)

- Defined in *File merge_integration.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**mergeLayerAintoLayerB**(**const** *Layer*<VoxelType> &*layer_A*, *Layer*<VoxelType> **layer_B*)

Merges layers, when the voxel or block size differs resampling occurs.

Template Function voxblox::mergeLayerAintoLayerB(const Layer<VoxelType>&, const Transformation&, Layer<VoxelType> *, bool)

Function Documentation

template <**typename** VoxelType>

void voxblox::**mergeLayerAintoLayerB**(**const** *Layer*<VoxelType> &*layer_A*, **const** *Transformation* &*T_B_A*, *Layer*<VoxelType> **layer_B*, bool *use_naive_method* = false)

Performs a 3D transformation on layer A before merging it.

See transformLayer for details

Template Function voxblox::mergeVoxelAIntoVoxelB(const VoxelType&, VoxelType *)

- Defined in *File voxel_utils.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**mergeVoxelAIntoVoxelB**(**const** VoxelType &*voxel_A*, VoxelType **voxel_B*)

Function voxblox::mergeVoxelAIntoVoxelB(const TsdfVoxel&, TsdfVoxel *)

Function Documentation

template <>
void voxblox::**mergeVoxelAIntoVoxelB**(**const** *TsdfVoxel* &*voxel_A*, *TsdfVoxel* **voxel_B*)

Function voxblox::mergeVoxelAIntoVoxelB(const EsdfVoxel&, EsdfVoxel *)

Function Documentation

template <>
void voxblox::**mergeVoxelAIntoVoxelB**(**const** *EsdfVoxel* &*voxel_A*, *EsdfVoxel* **voxel_B*)

Function voxblox::mergeVoxelAIntoVoxelB(const OccupancyVoxel&, OccupancyVoxel *)

Function Documentation

template <>
void voxblox::**mergeVoxelAIntoVoxelB**(**const** *OccupancyVoxel* &*voxel_A*, *OccupancyVoxel* **voxel_B*)

Template Function voxblox::naiveTransformLayer

- Defined in *File merge_integration.h*

Function Documentation

template <**typename** VoxelType>

void voxblox::**naiveTransformLayer**(**const** *Layer*<VoxelType> &*layer_in*, **const** *Transformation* &*T_out_in*, *Layer*<VoxelType> **layer_out*)

Similar to transformLayer in functionality, however the system only makes use of the forward transform and nearest neighbor interpolation.

This will result in artifacts and other issues in the result, however it should be several orders of magnitude faster.

Function voxblox::normalColorFromNormal

- Defined in *File mesh_vis.h*

Function Documentation

void voxblox::**normalColorFromNormal**(**const** *Point* &*normal*, std_msgs::ColorRGBA **color_msg*)

Function voxblox::outputMeshAsPly

Function Documentation

bool voxblox::**outputMeshAsPly**(**const** std::string &*filename*, **const** *Mesh* &*mesh*)

Function voxblox::outputMeshLayerAsPly(const std::string&, const MeshLayer&)

- Defined in *File mesh_ply.h*

Function Documentation

bool voxblox::**outputMeshLayerAsPly**(**const** std::string &*filename*, **const** *MeshLayer* &*mesh_layer*)

Default behaviour is to simplify the mesh.

Function voxblox::outputMeshLayerAsPly(const std::string&, const bool, const MeshLayer&)

Function Documentation

bool voxblox::**outputMeshLayerAsPly**(**const** std::string &*filename*, **const** bool *connected_mesh*, **const** *MeshLayer* &*mesh_layer*)

Parameters

- `connected_mesh`: if true veracities will be shared between triangles

Function voxblox::pointcloudToPclXYZ

- Defined in *File conversions.h*

Function Documentation

void voxblox::**pointcloudToPclXYZ**(**const** *Pointcloud* &*ptcloud*, pcl::PointCloud<pcl::PointXYZ> **ptcloud_pcl*)

Function voxblox::pointcloudToPclXYZI

- Defined in *File conversions.h*

Function Documentation

void voxblox::**pointcloudToPclXYZI**(**const** *Pointcloud* &*ptcloud*, **const** std::vector<float> &*intensities*, pcl::PointCloud<pcl::PointXYZI> **ptcloud_pcl*)

Function voxblox::pointcloudToPclXYZRGB

- Defined in *File conversions.h*

Function Documentation

void voxblox::**pointcloudToPclXYZRGB**(**const** *Pointcloud* &*ptcloud*, **const** *Colors* &*colors*, pcl::PointCloud<pcl::PointXYZRGB> **ptcloud_pcl*)

Function voxblox::probabilityFromLogOdds

- Defined in *File common.h*

Function Documentation

float voxblox::**probabilityFromLogOdds**(float *log_odds*)

Function voxblox::rainbowColorMap

Function Documentation

Color voxblox::**rainbowColorMap**(double *h*)

Maps an input h from a value between 0.0 and 1.0 into a rainbow.

Copied from OctomapProvider in octomap.

Function voxblox::randomColor

- Defined in *File color.h*

Function Documentation

Color voxblox::**randomColor**()

Function voxblox::recolorVoxbloxMeshMsgByIntensity

- Defined in *File intensity_vis.h*

Function Documentation

void voxblox::**recolorVoxbloxMeshMsgByIntensity**(**const** *Layer*<*IntensityVoxel*> &*intensity_layer*, **const** std::shared_ptr<*ColorMap*> &*color_map*, voxblox_msgs::Mesh **mesh_msg*)

Template Function voxblox::resampleLayer

- Defined in *File merge_integration.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**resampleLayer**(**const** *Layer*<VoxelType> &*layer_in*, *Layer*<VoxelType> **layer_out*)

copies the information stored in layer_in into layer_out resampling the data so that it fits the voxel and block size of the output layer

Template Function voxblox::serializeLayerAsMsg

Function Documentation

template <**typename** VoxelType>
void voxblox::**serializeLayerAsMsg**(**const** *Layer*<VoxelType> &*layer*, **const** bool *only_updated*, voxblox_msgs::Layer **msg*, **const** *MapDerializationAction* &*action* = *MapDerializationAction*::kUpdate)

Function voxblox::signum

- Defined in *File common.h*

Function Documentation

int voxblox::**signum**(*FloatingPoint x*)

Template Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, VoxelType *)

- Defined in *File layer_test_utils.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::test::**fillVoxelWithTestData**(size_t *x*, size_t *y*, size_t *z*, VoxelType **voxel*)

Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, TsdfVoxel *)

Function Documentation

template <>
void voxblox::test::**fillVoxelWithTestData**(size_t *x*, size_t *y*, size_t *z*, *TsdfVoxel* **voxel*)

Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, EsdfVoxel *)

Function Documentation

template <>
void voxblox::test::**fillVoxelWithTestData**(size_t *x*, size_t *y*, size_t *z*, *EsdfVoxel* **voxel*)

Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, OccupancyVoxel *)

Function Documentation

template <>
void voxblox::test::**fillVoxelWithTestData**(size_t *x*, size_t *y*, size_t *z*, *OccupancyVoxel* **voxel*)

Function voxblox::test::fillVoxelWithTestData(size_t, size_t, size_t, IntensityVoxel *)

Function Documentation

template <>
void voxblox::test::**fillVoxelWithTestData**(size_t *x*, size_t *y*, size_t *z*, *IntensityVoxel* **voxel*)

Template Function voxblox::test::SetUpTestLayer(const IndexElement, const IndexElement, Layer<VoxelType> *)

- Defined in *File layer_test_utils.h*

Function Documentation

template <typename VoxelType>
void voxblox::test::**SetUpTestLayer**(**const** *IndexElement* *block_volume_diameter*, **const** *IndexElement* *block_volume_offset*, *Layer*<VoxelType> **layer*)

Template Function voxblox::test::SetUpTestLayer(const IndexElement, Layer<VoxelType> *)

Function Documentation

template <typename VoxelType>
void voxblox::test::**SetUpTestLayer**(**const** *IndexElement* *block_volume_diameter*, *Layer*<VoxelType> **layer*)

Function voxblox::toConnectedPCLPolygonMesh

- Defined in *File mesh_pcl.h*

Function Documentation

void voxblox::**toConnectedPCLPolygonMesh**(**const** *MeshLayer* &*mesh_layer*, **const** std::string *frame_id*, pcl::PolygonMesh **polygon_mesh_ptr*)

Function voxblox::toPCLPolygonMesh

- Defined in *File mesh_pcl.h*

Function Documentation

void voxblox::toPCLPolygonMesh(const MeshLayer & mesh_layer, const std::string frame_id,

Function voxblox::toSimplifiedPCLPolygonMesh

- Defined in *File mesh_pcl.h*

Function Documentation

void voxblox::**toSimplifiedPCLPolygonMesh**(**const** *MeshLayer* &*mesh_layer*, **const** std::string *frame_id*, **const** *FloatingPoint* *vertex_proximity_threshold*, pcl::PolygonMesh **polygon_mesh_ptr*)

Template Function voxblox::transformLayer

- Defined in *File merge_integration.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::**transformLayer**(**const** *Layer*<VoxelType> &*layer_in*, **const** *Transformation* &*T_out_in*, *Layer*<VoxelType> **layer_out*)

Performs a 3D transform on the input layer and writes the results to the output layer.

During the transformation resampling occurs so that the voxel and block size of the input and output layer can differ.

Function voxblox::transformPointcloud

- Defined in *File common.h*

Function Documentation

void voxblox::**transformPointcloud**(**const** *Transformation* &*T_N_O*, **const** *Pointcloud* &*ptcloud*, *Pointcloud* **ptcloud_out*)

Template Function voxblox::utils::centerBlocksOfLayer

- Defined in *File layer_utils.h*

Function Documentation

template <**typename** VoxelType>
void voxblox::utils::**centerBlocksOfLayer**(*Layer*<VoxelType> **layer*, *Point* **new_layer_origin*)

This function will shift all the blocks such that the new grid origin will be close to the centroid of all allocated blocks.

The new_layer_origin is the origin of the new grid expressed in the old grids coordinate frame.

Template Function voxblox::utils::clearSphereAroundPoint

Function Documentation

template <**typename** VoxelType>
void voxblox::utils::**clearSphereAroundPoint**(**const** *Point* &*center*, **const** *FloatingPoint* *radius*, **const** *FloatingPoint* *max_distance_m*, *Layer*<VoxelType> **layer*)

Template Function voxblox::utils::computeMapBoundsFromLayer

Function Documentation

template <**typename** VoxelType>
void voxblox::utils::**computeMapBoundsFromLayer**(**const** voxblox::*Layer*<VoxelType> &*layer*, Eigen::Vector3d **lower_bound*, Eigen::Vector3d **upper_bound*)

Utility function to get map bounds from an arbitrary layer.

Only accurate to block level (i.e., outer bounds of allocated blocks).

Template Function voxblox::utils::computeVoxelError

- Defined in *File evaluation_utils.h*

Function Documentation

template <**typename** VoxelType>

VoxelEvaluationResult voxblox::utils::**computeVoxelError**(**const** VoxelType &*voxel_gt*, **const** VoxelType &*voxel_test*, **const** *VoxelEvaluationMode* *evaluation_mode*, *FloatingPoint* **error*)

Template Function voxblox::utils::evaluateLayersRmse(const Layer<VoxelType>&, const Layer<VoxelType>&, const VoxelEvaluationMode&, VoxelEvaluationDetails *, Layer<VoxelType> *)

Function Documentation

template <**typename** VoxelType>
FloatingPoint voxblox::utils::**evaluateLayersRmse**(**const** *Layer*<VoxelType> &*layer_gt*, **const** *Layer*<VoxelType> &*layer_test*, **const** *VoxelEvaluationMode* &*voxel_evaluation_mode*, *VoxelEvaluationDetails* **evaluation_result* = nullptr, *Layer*<VoxelType> **error_layer* = nullptr)

Evaluate a test layer vs a ground truth layer.

The comparison is symmetrical unless the VoxelEvaluationMode is set to ignore the voxels of one of the two layers behind the surface. The parameter 'evaluation_result' and 'error_layer' can be a nullptr.

Template Function voxblox::utils::evaluateLayersRmse(const Layer<VoxelType>&, const Layer<VoxelType>&)

Function Documentation

template <**typename** VoxelType>
FloatingPoint voxblox::utils::**evaluateLayersRmse**(**const** *Layer*<VoxelType> &*layer_gt*, **const** *Layer*<VoxelType> &*layer_test*)

Overload for convenient RMSE calculation.

Per default this function does not evaluate errors behind the test surface.

Template Function voxblox::utils::fillSphereAroundPoint

Function Documentation

template <**typename** VoxelType>
void voxblox::utils::**fillSphereAroundPoint**(**const** *Point* &*center*, **const** *FloatingPoint* *radius*, **const** *FloatingPoint* *max_distance_m*, *Layer*<VoxelType> **layer*)

Tools for manually editing a set of voxels.

Sets the values around a sphere to be artifically free or occupied, and marks them as hallucinated.

Template Function voxblox::utils::getAndAllocateSphereAroundPoint

Function Documentation

template <**typename** VoxelType>

void voxblox::utils::**getAndAllocateSphereAroundPoint**(**const** *Point* &*center*, *FloatingPoint* *radius*, *Layer*<VoxelType> **layer*, *HierarchicalIndexMap* **block_voxel_list*)

Gets the indices of all points around a sphere, and also allocates any blocks that don't already exist.

Template Function voxblox::utils::getColorIfValid(const VoxelType&, const FloatingPoint, Color *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::utils::**getColorIfValid**(**const** VoxelType &*voxel*, **const** *FloatingPoint* *min_weight*, *Color* **color*)

Function voxblox::utils::getColorIfValid(const TsdfVoxel&, const FloatingPoint, Color *)

Function Documentation

template <>
bool voxblox::utils::**getColorIfValid**(**const** *TsdfVoxel* &*voxel*, **const** *FloatingPoint* *min_weight*, *Color* **color*)

Function voxblox::utils::getColorIfValid(const EsdfVoxel&, const FloatingPoint, Color *)

Function Documentation

template <>
bool voxblox::utils::**getColorIfValid**(**const** *EsdfVoxel* &*voxel*, **const** *FloatingPoint*, *Color* **color*)

Template Function voxblox::utils::getSdfIfValid(const VoxelType&, const FloatingPoint, FloatingPoint *)

Function Documentation

template <**typename** VoxelType>
bool voxblox::utils::**getSdfIfValid**(**const** VoxelType &*voxel*, **const** *FloatingPoint* *min_weight*, *FloatingPoint* **sdf*)

Function voxblox::utils::getSdfIfValid(const TsdfVoxel&, const FloatingPoint, FloatingPoint *)

Function Documentation

template <>
bool voxblox::utils::**getSdfIfValid**(**const** *TsdfVoxel* &*voxel*, **const** *FloatingPoint* *min_weight*, *FloatingPoint* **sdf*)

Function voxblox::utils::getSdfIfValid(const EsdfVoxel&, const FloatingPoint, FloatingPoint *)

Function Documentation

template <>
bool voxblox::utils::**getSdfIfValid**(**const** *EsdfVoxel* &*voxel*, **const** *FloatingPoint*, *FloatingPoint* **sdf*)

Template Function voxblox::utils::getSphereAroundPoint

Function Documentation

template <typename VoxelType>
void voxblox::utils::**getSphereAroundPoint**(**const** *Layer*<VoxelType> &*layer*, **const** *Point* &*center*, *FloatingPoint* *radius*, *HierarchicalIndexMap* **block_voxel_list*)

Gets the indices of all points within the sphere.

Template Function voxblox::utils::getVoxelSdf(const VoxelType&)

- Defined in *File evaluation_utils.h*

Function Documentation

template <typename VoxelType>
FloatingPoint voxblox::utils::**getVoxelSdf**(**const** VoxelType &*voxel*)

Allow this class to be templated on all kinds of voxels.

Function voxblox::utils::getVoxelSdf(const TsdfVoxel&)

Function Documentation

template <>
FloatingPoint voxblox::utils::**getVoxelSdf**(**const** *TsdfVoxel* &*voxel*)

Function voxblox::utils::getVoxelSdf(const EsdfVoxel&)

Function Documentation

template <>
FloatingPoint voxblox::utils::**getVoxelSdf**(**const** *EsdfVoxel* &*voxel*)

Template Function voxblox::utils::isObservedVoxel(const VoxelType&)

Function Documentation

template <typename VoxelType>

bool voxblox::utils::**isObservedVoxel**(**const** VoxelType &*voxel*)
Returns true if the voxel has been observed.

Function voxblox::utils::isObservedVoxel(const TsdfVoxel&)

Function Documentation

template <>
bool voxblox::utils::**isObservedVoxel**(**const** *TsdfVoxel* &*voxel*)

Function voxblox::utils::isObservedVoxel(const EsdfVoxel&)

Function Documentation

template <>
bool voxblox::utils::**isObservedVoxel**(**const** *EsdfVoxel* &*voxel*)

Template Function voxblox::utils::isSameBlock

- Defined in *File layer_utils.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::utils::**isSameBlock**(**const** *Block*<VoxelType> &*block_A*, **const** *Block*<VoxelType> &*block_B*)

Template Function voxblox::utils::isSameLayer

- Defined in *File layer_utils.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::utils::**isSameLayer**(**const** *Layer*<VoxelType> &*layer_A*, **const** *Layer*<VoxelType> &*layer_B*)

Template Function voxblox::utils::isSameVoxel(const VoxelType&, const VoxelType&)

- Defined in *File layer_utils.h*

Function Documentation

template <**typename** VoxelType>
bool voxblox::utils::**isSameVoxel**(**const** VoxelType&, **const** VoxelType&)

Function voxblox::utils::isSameVoxel(const TsdfVoxel&, const TsdfVoxel&)

Function Documentation

template <>
bool voxblox::utils::**isSameVoxel**(**const** *TsdfVoxel* &*voxel_A*, **const** *TsdfVoxel* &*voxel_B*)

Function voxblox::utils::isSameVoxel(const EsdfVoxel&, const EsdfVoxel&)

Function Documentation

template <>
bool voxblox::utils::**isSameVoxel**(**const** *EsdfVoxel* &*voxel_A*, **const** *EsdfVoxel* &*voxel_B*)

Function voxblox::utils::isSameVoxel(const OccupancyVoxel&, const OccupancyVoxel&)

Function Documentation

template <>
bool voxblox::utils::**isSameVoxel**(**const** *OccupancyVoxel* &*voxel_A*, **const** *OccupancyVoxel* &*voxel_B*)

Function voxblox::utils::readProtoMsgCountToStream

Function Documentation

bool voxblox::utils::**readProtoMsgCountToStream**(std::fstream **stream_in*, uint32_t **message_count*, uint32_t **byte_offset*)

Function voxblox::utils::readProtoMsgFromStream

Function Documentation

bool voxblox::utils::**readProtoMsgFromStream**(std::fstream **stream_in*, google::protobuf::Message **message*, uint32_t **byte_offset*)

Template Function voxblox::utils::setVoxelSdf(const FloatingPoint, VoxelType *)

- Defined in *File evaluation_utils.h*

Function Documentation

template <typename VoxelType>
void voxblox::utils::**setVoxelSdf**(**const** *FloatingPoint sdf*, VoxelType **voxel*)

Function voxblox::utils::setVoxelSdf(const FloatingPoint, TsdfVoxel *)

Function Documentation

template <>
void voxblox::utils::**setVoxelSdf**(**const** *FloatingPoint sdf*, *TsdfVoxel* **voxel*)

Function voxblox::utils::setVoxelSdf(const FloatingPoint, EsdfVoxel *)

Function Documentation

template <>
void voxblox::utils::**setVoxelSdf**(**const** *FloatingPoint sdf*, *EsdfVoxel* **voxel*)

Template Function voxblox::utils::setVoxelWeight(const FloatingPoint, VoxelType *)

- Defined in *File evaluation_utils.h*

Function Documentation

template <typename VoxelType>
void voxblox::utils::**setVoxelWeight**(**const** *FloatingPoint weight*, VoxelType **voxel*)

Function voxblox::utils::setVoxelWeight(const FloatingPoint, TsdfVoxel *)

Function Documentation

template <>
void voxblox::utils::**setVoxelWeight**(**const** *FloatingPoint weight*, *TsdfVoxel* **voxel*)

Function voxblox::utils::setVoxelWeight(const FloatingPoint, EsdfVoxel *)

Function Documentation

template <>
void voxblox::utils::**setVoxelWeight**(**const** *FloatingPoint weight*, *EsdfVoxel* **voxel*)

Function voxblox::utils::writeProtoMsgCountToStream

Function Documentation

bool voxblox::utils::**writeProtoMsgCountToStream**(uint32_t *message_count*, std::fstream **stream_out*)

Function voxblox::utils::writeProtoMsgToStream

Function Documentation

bool voxblox::utils::**writeProtoMsgToStream**(**const** google::protobuf::Message &*message*, std::fstream **stream_out*)

Function voxblox::visualizeDistanceIntensityEsdfVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeDistanceIntensityEsdfVoxels**(**const** *EsdfVoxel* &*voxel*, **const** *Point*&, double **intensity*)

Function voxblox::visualizeDistanceIntensityEsdfVoxelsSlice

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeDistanceIntensityEsdfVoxelsSlice**(**const** *EsdfVoxel* &*voxel*, **const** *Point* &*coord*, unsigned int *free_plane_index*, *FloatingPoint* *free_plane_val*, *FloatingPoint* *voxel_size*, double **intensity*)

Function voxblox::visualizeDistanceIntensityTsdfVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeDistanceIntensityTsdfVoxels**(**const** *TsdfVoxel* &*voxel*, **const** *Point*&, double **intensity*)

Function voxblox::visualizeDistanceIntensityTsdfVoxelsNearSurface

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeDistanceIntensityTsdfVoxelsNearSurface**(**const** *TsdfVoxel* &*voxel*, **const** *Point*&, double *surface_distance*, double **intensity*)

Function voxblox::visualizeDistanceIntensityTsdfVoxelsSlice

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeDistanceIntensityTsdfVoxelsSlice**(**const** *TsdfVoxel* &*voxel*, **const** *Point* &*coord*, unsigned int *free_plane_index*, *FloatingPoint* *free_plane_val*, *FloatingPoint* *voxel_size*, double **intensity*)

Function voxblox::visualizeFreeEsdfVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeFreeEsdfVoxels**(**const** *EsdfVoxel* &*voxel*, **const** *Point*&, float *min_distance*, double **intensity*)

Function voxblox::visualizeIntensityVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeIntensityVoxels**(**const** *IntensityVoxel* &*voxel*, **const** *Point*&, double **intensity*)

Function voxblox::visualizeNearSurfaceTsdfVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeNearSurfaceTsdfVoxels**(**const** *TsdfVoxel* &*voxel*, **const** *Point*&, double *surface_distance*, *Color* *color)

Function voxblox::visualizeOccupiedOccupancyVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeOccupiedOccupancyVoxels**(**const** *OccupancyVoxel* &*voxel*, **const** *Point*&)

Function voxblox::visualizeOccupiedTsdfVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeOccupiedTsdfVoxels**(**const** *TsdfVoxel* &*voxel*, **const** *Point*&)

Function voxblox::visualizeTsdfVoxels

- Defined in *File ptcloud_vis.h*

Function Documentation

bool voxblox::**visualizeTsdfVoxels**(**const** *TsdfVoxel* &*voxel*, **const** *Point*&, *Color* *color)

10.3.5 Variables

Variable voxblox::kDefaultMaxIntensity

- Defined in *File tsdf_server.h*

Variable Documentation

constexpr float voxblox::**kDefaultMaxIntensity** = 100.0

Variable voxblox::kEpsilon

Variable Documentation

constexpr *FloatingPoint* voxblox::**kEpsilon** = 1e-6

Used for coordinates.

Variable voxblox::kFloatEpsilon

Variable Documentation

constexpr float voxblox::**kFloatEpsilon** = 1e-6
Used for weights.

Variable voxblox::kNumTsdfIntegratorTypes

- Defined in *File tsdf_integrator.h*

Variable Documentation

constexpr size_t voxblox::**kNumTsdfIntegratorTypes** = 3u

Variable voxblox::kTsdfIntegratorTypeNames

- Defined in *File tsdf_integrator.h*

Variable Documentation

"merged", "fast"}}]

Variable voxblox::kUnitCubeDiagonalLength

- Defined in *File merge_integration.h*

Variable Documentation

const *FloatingPoint* voxblox::**kUnitCubeDiagonalLength** = std::sqrt(3.0)

Variable voxblox::voxel_types::kEsdf

- Defined in *File voxel.h*

Variable Documentation

const std::string voxblox::voxel_types::**kEsdf** = "esdf"

Variable voxblox::voxel_types::kIntensity

- Defined in *File voxel.h*

Variable Documentation

const std::string voxblox::voxel_types::**kIntensity** = "intensity"

Variable voxblox::voxel_types::kNotSerializable

Variable Documentation

const std::string voxblox::voxel_types::**kNotSerializable** = "not_serializable"

Variable voxblox::voxel_types::kOccupancy

- Defined in *File voxel.h*

Variable Documentation

const std::string voxblox::voxel_types::**kOccupancy** = "occupancy"

Variable voxblox::voxel_types::kTsdf

- Defined in *File voxel.h*

Variable Documentation

const std::string voxblox::voxel_types::**kTsdf** = "tsdf"

10.3.6 Typedefs

Typedef voxblox::AlignedDeque

Typedef Documentation

using voxblox::AlignedDeque = typedef std::deque<Type, Eigen::aligned_allocator<Type>>

Typedef voxblox::AlignedLayerAndErrorLayer

- Defined in *File merge_integration.h*

Typedef Documentation

typedef std::pair<voxblox::*Layer*<voxblox::*TsdfVoxel*>::Ptr, voxblox::*Layer*<voxblox::*TsdfVoxel*>::Ptr> voxblox::**AlignedLa**

Typedef voxblox::AlignedLayerAndErrorLayers

- Defined in *File merge_integration.h*

Typedef Documentation

typedef std::vector<*AlignedLayerAndErrorLayer*> voxblox::**AlignedLayerAndErrorLayers**

Typedef voxblox::AlignedList

- Defined in *File common.h*

Typedef Documentation

```
using voxblox::AlignedList = typedef std::list<Type, Eigen::aligned_allocator<Type>>
```

Typedef voxblox::AlignedQueue

Typedef Documentation

```
using voxblox::AlignedQueue = typedef std::queue<Type, AlignedDeque<Type>>
```

Typedef voxblox::AlignedStack

- Defined in *File common.h*

Typedef Documentation

```
using voxblox::AlignedStack = typedef std::stack<Type, AlignedDeque<Type>>
```

Typedef voxblox::AlignedVector

Typedef Documentation

```
using voxblox::AlignedVector = typedef std::vector<Type, Eigen::aligned_allocator<Type>>
```

Typedef voxblox::AnyIndex

Typedef Documentation

typedef Eigen::Matrix<*IndexElement*, 3, 1> `voxblox::`**AnyIndex**

Typedef voxblox::BlockIndex

Typedef Documentation

typedef *AnyIndex* `voxblox::`**BlockIndex**

Typedef voxblox::BlockIndexList

Typedef Documentation

typedef *IndexVector* `voxblox::`**BlockIndexList**

Typedef voxblox::Colors

Typedef Documentation

typedef AlignedVector<*Color*> voxblox::**Colors**

Typedef voxblox::FloatingPoint

Typedef Documentation

typedef float voxblox::**FloatingPoint**

Typedef voxblox::GlobalIndex

Typedef Documentation

typedef *LongIndex* voxblox::**GlobalIndex**

Typedef voxblox::GlobalIndexVector

- Defined in *File common.h*

Typedef Documentation

typedef *LongIndexVector* voxblox::**GlobalIndexVector**

Typedef voxblox::HierarchicalIndex

- Defined in *File block_hash.h*

Typedef Documentation

typedef *HierarchicalIndexMap*::value_type voxblox::**HierarchicalIndex**

Typedef voxblox::HierarchicalIndexMap

Typedef Documentation

typedef *AnyIndexHashMapType*<*IndexVector*>::type voxblox::**HierarchicalIndexMap**

Typedef voxblox::HierarchicalIndexSet

- Defined in *File block_hash.h*

Typedef Documentation

typedef *AnyIndexHashMapType*<*IndexSet*>::type voxblox::**HierarchicalIndexSet**

Typedef voxblox::IndexElement

Typedef Documentation

typedef int voxblox::**IndexElement**

Typedef voxblox::IndexSet

Typedef Documentation

typedef std::unordered_set<*AnyIndex*, *AnyIndexHash*, std::equal_to<*AnyIndex*>, Eigen::aligned_allocator<*AnyIndex*>> voxblox:

Typedef voxblox::IndexVector

- Defined in *File common.h*

Typedef Documentation

typedef AlignedVector<*AnyIndex*> voxblox::**IndexVector**

Typedef voxblox::InterpIndexes

Typedef Documentation

typedef Eigen::Array<*IndexElement*, 3, 8> voxblox::**InterpIndexes**

Typedef voxblox::InterpTable

Typedef Documentation

typedef Eigen::Matrix<*FloatingPoint*, 8, 8> voxblox::**InterpTable**

Typedef voxblox::InterpVector

Typedef Documentation

typedef Eigen::Matrix<*FloatingPoint*, 1, 8> voxblox::**InterpVector**

Typedef voxblox::Label

- Defined in *File common.h*

Typedef Documentation

typedef uint32_t `voxblox::`**Label**

Typedef voxblox::LabelConfidence

- Defined in *File common.h*

Typedef Documentation

typedef uint32_t `voxblox::`**LabelConfidence**

Typedef voxblox::Labels

- Defined in *File common.h*

Typedef Documentation

typedef AlignedVector<*Label*> `voxblox::`**Labels**

Typedef voxblox::LongIndex

Typedef Documentation

typedef Eigen::Matrix<*LongIndexElement*, 3, 1> `voxblox::`**LongIndex**

Typedef voxblox::LongIndexElement

- Defined in *File common.h*

Typedef Documentation

typedef int64_t `voxblox::`**LongIndexElement**

Typedef voxblox::LongIndexSet

Typedef Documentation

typedef std::unordered_set<*LongIndex*, *LongIndexHash*, std::equal_to<*LongIndex*>, Eigen::aligned_allocator<*LongIndex*>> `voxbl`

Typedef voxblox::LongIndexVector

- Defined in *File common.h*

Typedef Documentation

typedef AlignedVector<*LongIndex*> voxblox::**LongIndexVector**

Typedef voxblox::Point

Typedef Documentation

typedef Eigen::Matrix<*FloatingPoint*, 3, 1> voxblox::**Point**

Typedef voxblox::Pointcloud

Typedef Documentation

typedef AlignedVector<*Point*> voxblox::**Pointcloud**

Typedef voxblox::PointsMatrix

Typedef Documentation

typedef Eigen::Matrix<*FloatingPoint*, 3, Eigen::Dynamic> voxblox::**PointsMatrix**

Typedef voxblox::Quaternion

Typedef Documentation

typedef kindr::minimal::RotationQuaternionTemplate<*FloatingPoint*>::Implementation voxblox::**Quaternion**

Typedef voxblox::Ray

Typedef Documentation

typedef Eigen::Matrix<*FloatingPoint*, 3, 1> voxblox::**Ray**

Typedef voxblox::Rotation

Typedef Documentation

typedef kindr::minimal::RotationQuaternionTemplate<*FloatingPoint*> voxblox::**Rotation**

Typedef voxblox::ShouldVisualizeVoxelColorFunctionType

- Defined in *File ptcloud_vis.h*

Typedef Documentation

using voxblox::ShouldVisualizeVoxelColorFunctionType = typedef std::function<bool(const Vo

Shortcut the placeholders namespace, since otherwise it chooses the boost placeholders which are in the global namespace (thanks boost!).

This is a fancy alias to be able to template functions.

Typedef voxblox::ShouldVisualizeVoxelFunctionType

- Defined in *File ptcloud_vis.h*

Typedef Documentation

using voxblox::ShouldVisualizeVoxelFunctionType = typedef std::function<bool(const VoxelTyp

For boolean checks either a voxel is visualized or it is not.

This is used for occupancy bricks, for instance.

Typedef voxblox::ShouldVisualizeVoxelIntensityFunctionType

- Defined in *File ptcloud_vis.h*

Typedef Documentation

using voxblox::ShouldVisualizeVoxelIntensityFunctionType = typedef std::function<bool(cons

For intensities values, such as distances, which are mapped to a color only by the subscriber.

Typedef voxblox::SignedIndex

Typedef Documentation

typedef *AnyIndex* voxblox::**SignedIndex**

Typedef voxblox::SquareMatrix

Typedef Documentation

using voxblox::SquareMatrix = typedef Eigen::Matrix<FloatingPoint, size, size>

Typedef voxblox::timing::DebugTimer

- Defined in *File timing.h*

Typedef Documentation

typedef *DummyTimer* voxblox::timing::**DebugTimer**

Typedef voxblox::Transformation

Typedef Documentation

typedef kindr::minimal::QuatTransformationTemplate<*FloatingPoint*> voxblox::**Transformation**

Typedef voxblox::Triangle

Typedef Documentation

typedef Eigen::Matrix<*FloatingPoint*, 3, 3> voxblox::**Triangle**

Typedef voxblox::TriangleVector

Typedef Documentation

typedef AlignedVector<*Triangle*> voxblox::**TriangleVector**

Typedef voxblox::VertexIndex

Typedef Documentation

typedef size_t voxblox::**VertexIndex**

Typedef voxblox::VertexIndexList

Typedef Documentation

typedef AlignedVector<*VertexIndex*> voxblox::**VertexIndexList**

Typedef voxblox::VoxelIndex

Typedef Documentation

typedef *AnyIndex* voxblox::**VoxelIndex**

Typedef voxblox::VoxelIndexList

- Defined in *File common.h*

Typedef Documentation

typedef *IndexVector* voxblox::**VoxelIndexList**

Typedef voxblox::VoxelKey

Typedef Documentation

typedef std::pair<*BlockIndex*, *VoxelIndex*> voxblox::**VoxelKey**

10.3.7 Directories

Directory include

Parent directory (voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include

Subdirectories

- *Directory voxblox*

Directory voxblox

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox

Subdirectories

- *Directory alignment*
- *Directory core*
- *Directory integrator*
- *Directory interpolator*
- *Directory io*
- *Directory mesh*
- *Directory simulation*
- *Directory test*
- *Directory utils*

Directory alignment

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/alignment

Directory core

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/core

Directory integrator

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/integrator

Directory interpolator

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/interpolator

Directory io

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/io

Directory mesh

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/mesh

Directory simulation

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/simulation

Directory test

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/test

Directory utils

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox/include/voxblox/utils

Directory include

Parent directory (voxblox_ros)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox_ros/include

Subdirectories

- *Directory ros*

Directory ros

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox_ros/include)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox_ros/include/voxblox_ros

Directory include

Parent directory (voxblox_rviz_plugin)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox_rviz_plugin/include

Subdirectories

- *Directory plugin*

Directory plugin

Parent directory (/home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox_rviz_plugin/include)

Directory path: /home/docs/checkouts/readthedocs.org/user_builds/voxblox/checkouts/latest/voxblox_rviz_plugin/include/voxblox_rviz_plugin

Directory voxblox

Directory path: voxblox

Subdirectories

- *Directory include*

Directory ros

Directory path: voxblox_ros

Subdirectories

- *Directory include*

Directory plugin

Directory path: voxblox_rviz_plugin

Subdirectories

- *Directory include*

10.3.8 Files

File approx_hash_array.h

Contents

- *Definition (voxblox/include/voxblox/utils/approx_hash_array.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/utils/approx_hash_array.h)

Program Listing for File approx_hash_array.h

Return to documentation for file (voxblox/include/voxblox/utils/approx_hash_array.h)

```
#ifndef VOXBLOX_UTILS_APPROX_HASH_ARRAY_H_
#define VOXBLOX_UTILS_APPROX_HASH_ARRAY_H_

#include <atomic>
#include <limits>
#include <vector>
#include "voxblox/core/common.h"

namespace voxblox {

template <size_t unmasked_bits, typename StoredElement, typename IndexType,
          typename IndexTypeHasher>
class ApproxHashArray {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  StoredElement& get(const size_t& hash) {
    return pseudo_map_[hash & bit_mask_];
  }

  StoredElement& get(const IndexType& index, size_t* hash) {
    DCHECK(hash);
    *hash = hasher_(index);
    return get(*hash);
  }

  StoredElement& get(const IndexType& index) {
    size_t hash = hasher_(index);
    return get(hash);
  }

 private:
  static constexpr size_t pseudo_map_size_ = (1 << unmasked_bits);
  static constexpr size_t bit_mask_ = (1 << unmasked_bits) - 1;

  std::array<StoredElement, pseudo_map_size_> pseudo_map_;
  IndexTypeHasher hasher_;
};

template <size_t unmasked_bits, size_t full_reset_threshold, typename IndexType,
          typename IndexTypeHasher>
class ApproxHashSet {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  ApproxHashSet() : offset_(0), pseudo_set_(pseudo_set_size_) {
    for (std::atomic<size_t>& value : pseudo_set_) {
      value.store(0, std::memory_order_relaxed);
    }
    // The array used for storing values is initialized with zeros. However, the
    // zeroth bin can actually store the 0 hash. Because of this to prevent a
```

(continues on next page)

(continued from previous page)

```
  // false positive on looking up a 0 hash this bin needs to initially store a
  // different value.
  pseudo_set_[offset_].store(std::numeric_limits<size_t>::max());
}

inline bool isHashCurrentlyPresent(const size_t& hash) {
  const size_t array_index = (hash & bit_mask_) + offset_;

  return (pseudo_set_[array_index].load(std::memory_order_relaxed) == hash);
}

inline bool isHashCurrentlyPresent(const IndexType& index, size_t* hash) {
  DCHECK(hash);
  *hash = hasher_(index);
  return isHashCurrentlyPresent(*hash);
}

inline bool isHashCurrentlyPresent(const IndexType& index) {
  size_t hash = hasher_(index);
  return isHashCurrentlyPresent(hash);
}
inline bool replaceHash(const size_t& hash) {
  const size_t array_index = (hash & bit_mask_) + offset_;

  if (pseudo_set_[array_index].load(std::memory_order_relaxed) == hash) {
    return false;
  } else {
    pseudo_set_[array_index].store(hash, std::memory_order_relaxed);
    return true;
  }
}

inline bool replaceHash(const IndexType& index, size_t* hash) {
  DCHECK(hash);
  *hash = hasher_(index);
  return replaceHash(*hash);
}

inline bool replaceHash(const IndexType& index) {
  const size_t hash = hasher_(index);
  return replaceHash(hash);
}

void resetApproxSet() {
  if (++offset_ >= full_reset_threshold) {
    for (std::atomic<size_t>& value : pseudo_set_) {
      value.store(0, std::memory_order_relaxed);
    }
    offset_ = 0;

    // The array used for storing values is initialized with zeros. However,
    // the zeroth bin can actually store the 0 hash. Because of this to
    // prevent a false positive on looking up a 0 hash this bin needs to
    // initially store a different value.
    pseudo_set_[offset_].store(std::numeric_limits<size_t>::max());
  }
}
```

(continues on next page)

(continued from previous page)

```
 private:
  static constexpr size_t pseudo_set_size_ =
      (1 << unmasked_bits) + full_reset_threshold;
  static constexpr size_t bit_mask_ = (1 << unmasked_bits) - 1;

  size_t offset_;
  std::vector<std::atomic<size_t>> pseudo_set_;

  IndexTypeHasher hasher_;
};
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_APPROX_HASH_ARRAY_H_
```

Includes

- atomic
- limits
- vector
- voxblox/core/common.h (*File common.h*)

Included By

- *File icp.h*
- *File tsdf_integrator.h*

Namespaces

- *Namespace voxblox*

Classes

- *Template Class ApproxHashArray*
- *Template Class ApproxHashSet*

File block.h

Contents

- *Definition (voxblox/include/voxblox/core/block.h)*
- *Includes*
- *Included By*

- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/core/block.h`)

Program Listing for File block.h

Return to documentation for file (`voxblox/include/voxblox/core/block.h`)

```
#ifndef VOXBLOX_CORE_BLOCK_H_
#define VOXBLOX_CORE_BLOCK_H_

#include <algorithm>
#include <atomic>
#include <memory>
#include <vector>

#include "./Block.pb.h"
#include "voxblox/core/common.h"

namespace voxblox {

template <typename VoxelType>
class Block {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<Block<VoxelType> > Ptr;
  typedef std::shared_ptr<const Block<VoxelType> > ConstPtr;

  Block(size_t voxels_per_side, FloatingPoint voxel_size, const Point& origin)
      : has_data_(false),
        voxels_per_side_(voxels_per_side),
        voxel_size_(voxel_size),
        origin_(origin),
        updated_(false) {
    num_voxels_ = voxels_per_side_ * voxels_per_side_ * voxels_per_side_;
    voxel_size_inv_ = 1.0 / voxel_size_;
    block_size_ = voxels_per_side_ * voxel_size_;
    block_size_inv_ = 1.0 / block_size_;
    voxels_.reset(new VoxelType[num_voxels_]);
  }

  explicit Block(const BlockProto& proto);

  ~Block() {}

  // Index calculations.
  inline size_t computeLinearIndexFromVoxelIndex(
      const VoxelIndex& index) const {
    size_t linear_index = static_cast<size_t>(
        index.x() +
        voxels_per_side_ * (index.y() + index.z() * voxels_per_side_));
```

(continues on next page)

(continued from previous page)

```
  DCHECK(index.x() >= 0 && index.x() < static_cast<int>(voxels_per_side_));
  DCHECK(index.y() >= 0 && index.y() < static_cast<int>(voxels_per_side_));
  DCHECK(index.z() >= 0 && index.z() < static_cast<int>(voxels_per_side_));

  DCHECK_LT(linear_index,
            voxels_per_side_ * voxels_per_side_ * voxels_per_side_);
  DCHECK_GE(linear_index, 0u);
  return linear_index;
}

inline VoxelIndex computeTruncatedVoxelIndexFromCoordinates(
    const Point& coords) const {
  const IndexElement max_value = voxels_per_side_ - 1;
  VoxelIndex voxel_index =
      getGridIndexFromPoint<VoxelIndex>(coords - origin_, voxel_size_inv_);
  // check is needed as getGridIndexFromPoint gives results that have a tiny
  // chance of being outside the valid voxel range.
  return VoxelIndex(std::max(std::min(voxel_index.x(), max_value), 0),
                    std::max(std::min(voxel_index.y(), max_value), 0),
                    std::max(std::min(voxel_index.z(), max_value), 0));
}

inline VoxelIndex computeVoxelIndexFromCoordinates(
    const Point& coords) const {
  VoxelIndex voxel_index =
      getGridIndexFromPoint<VoxelIndex>(coords - origin_, voxel_size_inv_);
  return voxel_index;
}

inline size_t computeLinearIndexFromCoordinates(const Point& coords) const {
  return computeLinearIndexFromVoxelIndex(
      computeTruncatedVoxelIndexFromCoordinates(coords));
}

inline Point computeCoordinatesFromLinearIndex(size_t linear_index) const {
  return computeCoordinatesFromVoxelIndex(
      computeVoxelIndexFromLinearIndex(linear_index));
}

inline Point computeCoordinatesFromVoxelIndex(const VoxelIndex& index) const {
  return origin_ + getCenterPointFromGridIndex(index, voxel_size_);
}

inline VoxelIndex computeVoxelIndexFromLinearIndex(
    size_t linear_index) const {
  int rem = linear_index;
  VoxelIndex result;
  std::div_t div_temp = std::div(rem, voxels_per_side_ * voxels_per_side_);
  rem = div_temp.rem;
  result.z() = div_temp.quot;
  div_temp = std::div(rem, voxels_per_side_);
  result.y() = div_temp.quot;
  result.x() = div_temp.rem;
  return result;
}

inline const VoxelType& getVoxelByLinearIndex(size_t index) const {
```

(continues on next page)

(continued from previous page)

```
  return voxels_[index];
}

inline const VoxelType& getVoxelByVoxelIndex(const VoxelIndex& index) const {
  return voxels_[computeLinearIndexFromVoxelIndex(index)];
}

inline const VoxelType& getVoxelByCoordinates(const Point& coords) const {
  return voxels_[computeLinearIndexFromCoordinates(coords)];
}

inline VoxelType& getVoxelByCoordinates(const Point& coords) {
  return voxels_[computeLinearIndexFromCoordinates(coords)];
}

inline VoxelType* getVoxelPtrByCoordinates(const Point& coords) {
  return &voxels_[computeLinearIndexFromCoordinates(coords)];
}

inline const VoxelType* getVoxelPtrByCoordinates(const Point& coords) const {
  return &voxels_[computeLinearIndexFromCoordinates(coords)];
}

inline VoxelType& getVoxelByLinearIndex(size_t index) {
  DCHECK_LT(index, num_voxels_);
  return voxels_[index];
}

inline VoxelType& getVoxelByVoxelIndex(const VoxelIndex& index) {
  return voxels_[computeLinearIndexFromVoxelIndex(index)];
}

inline bool isValidVoxelIndex(const VoxelIndex& index) const {
  if (index.x() < 0 ||
      index.x() >= static_cast<IndexElement>(voxels_per_side_)) {
    return false;
  }
  if (index.y() < 0 ||
      index.y() >= static_cast<IndexElement>(voxels_per_side_)) {
    return false;
  }
  if (index.z() < 0 ||
      index.z() >= static_cast<IndexElement>(voxels_per_side_)) {
    return false;
  }
  return true;
}

inline bool isValidLinearIndex(size_t index) const {
  if (index < 0 || index >= num_voxels_) {
    return false;
  }
  return true;
}

BlockIndex block_index() const {
  return getGridIndexFromOriginPoint<BlockIndex>(origin_, block_size_inv_);
```

(continues on next page)

(continued from previous page)

```
  }

  // Basic function accessors.
  size_t voxels_per_side() const { return voxels_per_side_; }
  FloatingPoint voxel_size() const { return voxel_size_; }
  FloatingPoint voxel_size_inv() const { return voxel_size_inv_; }
  size_t num_voxels() const { return num_voxels_; }
  Point origin() const { return origin_; }
  void setOrigin(const Point& new_origin) { origin_ = new_origin; }
  FloatingPoint block_size() const { return block_size_; }

  bool has_data() const { return has_data_; }
  bool updated() const { return updated_; }

  std::atomic<bool>& updated() { return updated_; }
  bool& has_data() { return has_data_; }

  void set_updated(bool updated) { updated_ = updated; }
  void set_has_data(bool has_data) { has_data_ = has_data; }

  // Serialization.
  void getProto(BlockProto* proto) const;
  void serializeToIntegers(std::vector<uint32_t>* data) const;
  void deserializeFromIntegers(const std::vector<uint32_t>& data);

  void mergeBlock(const Block<VoxelType>& other_block);

  size_t getMemorySize() const;

 protected:
  std::unique_ptr<VoxelType[]> voxels_;

  // Derived, cached parameters.
  size_t num_voxels_;

  bool has_data_;

 private:
  void deserializeProto(const BlockProto& proto);
  void serializeProto(BlockProto* proto) const;

  // Base parameters.
  const size_t voxels_per_side_;
  const FloatingPoint voxel_size_;
  Point origin_;

  // Derived, cached parameters.
  FloatingPoint voxel_size_inv_;
  FloatingPoint block_size_;
  FloatingPoint block_size_inv_;

  std::atomic<bool> updated_;
};

}  // namespace voxblox

#include "voxblox/core/block_inl.h"
```

(continues on next page)

(continued from previous page)

```
#endif  // VOXBLOX_CORE_BLOCK_H_
```

Includes

- ./Block.pb.h
- algorithm
- atomic
- memory
- vector
- voxblox/core/block_inl.h (*File block_inl.h*)
- voxblox/core/common.h (*File common.h*)

Included By

- *File layer.h*
- *File layer_inl.h*
- *File simulation_world_inl.h*
- *File distance_utils.h*

Namespaces

- *Namespace voxblox*

Classes

- *Template Class Block*

File block_hash.h

Contents

- *Definition (voxblox/include/voxblox/core/block_hash.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/core/block_hash.h)

Program Listing for File block_hash.h

Return to documentation for file (voxblox/include/voxblox/core/block_hash.h)

```
#ifndef VOXBLOX_CORE_BLOCK_HASH_H_
#define VOXBLOX_CORE_BLOCK_HASH_H_

#include <functional>
#include <unordered_map>
#include <unordered_set>
#include <utility>

#include <Eigen/Core>

#include "voxblox/core/common.h"

namespace voxblox {

struct AnyIndexHash {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  static constexpr size_t sl = 17191;
  static constexpr size_t sl2 = sl * sl;

  std::size_t operator()(const AnyIndex& index) const {
    return static_cast<unsigned int>(index.x() + index.y() * sl +
                                     index.z() * sl2);
  }
};

template <typename ValueType>
struct AnyIndexHashMapType {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::unordered_map<
      AnyIndex, ValueType, AnyIndexHash, std::equal_to<AnyIndex>,
      Eigen::aligned_allocator<std::pair<const AnyIndex, ValueType> > >
      type;
};

typedef std::unordered_set<AnyIndex, AnyIndexHash, std::equal_to<AnyIndex>,
                           Eigen::aligned_allocator<AnyIndex> >
    IndexSet;

typedef typename AnyIndexHashMapType<IndexVector>::type HierarchicalIndexMap;

typedef typename AnyIndexHashMapType<IndexSet>::type HierarchicalIndexSet;

typedef typename HierarchicalIndexMap::value_type HierarchicalIndex;

struct LongIndexHash {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  static constexpr size_t sl = 17191;
  static constexpr size_t sl2 = sl * sl;
```

(continues on next page)

(continued from previous page)

```
  std::size_t operator()(const LongIndex& index) const {
    return static_cast<unsigned int>(index.x() + index.y() * sl +
                                     index.z() * sl2);
  }
};

template <typename ValueType>
struct LongIndexHashMapType {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::unordered_map<
      LongIndex, ValueType, LongIndexHash, std::equal_to<LongIndex>,
      Eigen::aligned_allocator<std::pair<const LongIndex, ValueType> > >
      type;
};

typedef std::unordered_set<LongIndex, LongIndexHash, std::equal_to<LongIndex>,
                           Eigen::aligned_allocator<LongIndex> >
    LongIndexSet;

}  // namespace voxblox

#endif  // VOXBLOX_CORE_BLOCK_HASH_H_
```

Includes

- Eigen/Core
- functional
- unordered_map
- unordered_set
- utility
- voxblox/core/common.h (*File common.h*)

Included By

- *File icp.h*
- *File layer.h*
- *File integrator_utils.h*
- *File occupancy_integrator.h*
- *File tsdf_integrator.h*
- *File mesh_layer.h*
- *File mesh_utils.h*
- *File voxblox_mesh_visual.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct AnyIndexHash*
- *Template Struct AnyIndexHashMapType*
- *Struct LongIndexHash*
- *Template Struct LongIndexHashMapType*

File block_inl.h

Contents

- *Definition (voxblox/include/voxblox/core/block_inl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/core/block_inl.h)

Program Listing for File block_inl.h

Return to documentation for file (voxblox/include/voxblox/core/block_inl.h)

```
#ifndef VOXBLOX_CORE_BLOCK_INL_H_
#define VOXBLOX_CORE_BLOCK_INL_H_

#include <vector>

#include "./Block.pb.h"
#include "voxblox/utils/voxel_utils.h"

namespace voxblox {

template <typename VoxelType>
Block<VoxelType>::Block(const BlockProto& proto)
    : Block(proto.voxels_per_side(), proto.voxel_size(),
            Point(proto.origin_x(), proto.origin_y(), proto.origin_z())) {
  has_data_ = proto.has_data();

  // Convert the data into a vector of integers.
  std::vector<uint32_t> data;
  data.reserve(proto.voxel_data_size());

  for (uint32_t word : proto.voxel_data()) {
```

(continues on next page)

(continued from previous page)

```
    data.push_back(word);
  }

  deserializeFromIntegers(data);
}

template <typename VoxelType>
void Block<VoxelType>::getProto(BlockProto* proto) const {
  CHECK_NOTNULL(proto);

  proto->set_voxels_per_side(voxels_per_side_);
  proto->set_voxel_size(voxel_size_);

  proto->set_origin_x(origin_.x());
  proto->set_origin_y(origin_.y());
  proto->set_origin_z(origin_.z());

  proto->set_has_data(has_data_);

  std::vector<uint32_t> data;
  serializeToIntegers(&data);
  // Not quite actually a word since we're in a 64-bit age now, but whatever.
  for (uint32_t word : data) {
    proto->add_voxel_data(word);
  }
}

template <typename VoxelType>
void Block<VoxelType>::mergeBlock(const Block<VoxelType>& other_block) {
  CHECK_EQ(other_block.voxel_size(), voxel_size());
  CHECK_EQ(other_block.voxels_per_side(), voxels_per_side());

  if (!other_block.has_data()) {
    return;
  } else {
    has_data() = true;
    updated() = true;

    for (IndexElement voxel_idx = 0;
         voxel_idx < static_cast<IndexElement>(num_voxels()); ++voxel_idx) {
      mergeVoxelAIntoVoxelB<VoxelType>(
          other_block.getVoxelByLinearIndex(voxel_idx),
          &(getVoxelByLinearIndex(voxel_idx)));
    }
  }
}

template <typename VoxelType>
size_t Block<VoxelType>::getMemorySize() const {
  size_t size = 0u;

  // Calculate size of members
  size += sizeof(voxels_per_side_);
  size += sizeof(voxel_size_);
  size += sizeof(origin_);
  size += sizeof(num_voxels_);
  size += sizeof(voxel_size_inv_);
```

(continues on next page)

(continued from previous page)

```
  size += sizeof(block_size_);

  size += sizeof(has_data_);
  size += sizeof(updated_);

  if (num_voxels_ > 0u) {
    size += (num_voxels_ * sizeof(voxels_[0]));
  }
  return size;
}

}  // namespace voxblox

#endif  // VOXBLOX_CORE_BLOCK_INL_H_
```

Includes

- ./Block.pb.h
- vector
- voxblox/utils/voxel_utils.h (*File voxel_utils.h*)

Included By

- *File block.h*

Namespaces

- *Namespace voxblox*

File bucket_queue.h

Contents

- *Definition (voxblox/include/voxblox/utils/bucket_queue.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/utils/bucket_queue.h)

Program Listing for File bucket_queue.h

Return to documentation for file (voxblox/include/voxblox/utils/bucket_queue.h)

```
#ifndef VOXBLOX_UTILS_BUCKET_QUEUE_H_
#define VOXBLOX_UTILS_BUCKET_QUEUE_H_

#include <glog/logging.h>
#include <deque>
#include <queue>
#include <vector>

#include "voxblox/core/common.h"

namespace voxblox {
template <typename T>
class BucketQueue {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  BucketQueue() : last_bucket_index_(0) {}
  explicit BucketQueue(int num_buckets, double max_val)
      : num_buckets_(num_buckets),
        max_val_(max_val),
        last_bucket_index_(0),
        num_elements_(0) {
    buckets_.resize(num_buckets_);
  }

  void setNumBuckets(int num_buckets, double max_val) {
    max_val_ = max_val;
    num_buckets_ = num_buckets;
    buckets_.clear();
    buckets_.resize(num_buckets_);
    num_elements_ = 0;
  }

  void push(const T& key, double value) {
    CHECK_NE(num_buckets_, 0);
    if (value > max_val_) {
      value = max_val_;
    }
    int bucket_index =
        std::floor(std::abs(value) / max_val_ * (num_buckets_ - 1));
    if (bucket_index >= num_buckets_) {
      bucket_index = num_buckets_ - 1;
    }
    if (bucket_index < last_bucket_index_) {
      last_bucket_index_ = bucket_index;
    }
    buckets_[bucket_index].push(key);
    num_elements_++;
  }

  void pop() {
    if (empty()) {
      return;
    }
    while (buckets_[last_bucket_index_].empty() &&
           last_bucket_index_ < num_buckets_) {
      last_bucket_index_++;
```

(continues on next page)

(continued from previous page)

```
    }
    if (last_bucket_index_ < num_buckets_) {
      buckets_[last_bucket_index_].pop();
      num_elements_--;
    }
  }

  T front() {
    CHECK_NE(num_buckets_, 0);
    CHECK(!empty());
    while (buckets_[last_bucket_index_].empty() &&
           last_bucket_index_ < num_buckets_) {
      last_bucket_index_++;
    }
    return buckets_[last_bucket_index_].front();
  }

  bool empty() { return num_elements_ == 0; }

  void clear() {
    buckets_.clear();
    buckets_.resize(num_buckets_);
    last_bucket_index_ = 0;
    num_elements_ = 0;
  }

 private:
  int num_buckets_;
  double max_val_;
  voxblox::AlignedVector<voxblox::AlignedQueue<T>> buckets_;

  int last_bucket_index_;
  size_t num_elements_;
};
}  // namespace voxblox
#endif  // VOXBLOX_UTILS_BUCKET_QUEUE_H_
```

Includes

- deque
- glog/logging.h
- queue (*File bucket_queue.h*)
- vector
- voxblox/core/common.h (*File common.h*)

Included By

- *File esdf_integrator.h*
- *File esdf_occ_integrator.h*

Namespaces

- *Namespace voxblox*

Classes

- *Template Class BucketQueue*

File camera_model.h

Contents

- *Definition (voxblox/include/voxblox/utils/camera_model.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/utils/camera_model.h)

Program Listing for File camera_model.h

Return to documentation for file (voxblox/include/voxblox/utils/camera_model.h)

```
#ifndef VOXBLOX_UTILS_CAMERA_MODEL_H_
#define VOXBLOX_UTILS_CAMERA_MODEL_H_

#include <vector>

#include <glog/logging.h>
#include <kindr/minimal/quat-transformation.h>
#include <Eigen/Core>

#include "voxblox/core/common.h"

namespace voxblox {

class Plane {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  Plane() : normal_(Point::Identity()), distance_(0) {}
  virtual ~Plane() {}

  void setFromPoints(const Point& p1, const Point& p2, const Point& p3);
  void setFromDistanceNormal(const Point& normal, double distance);

  bool isPointInside(const Point& point) const;

  Point normal() const { return normal_; }
```

(continues on next page)

(continued from previous page)

```
  double distance() const { return distance_; }

 private:
  Point normal_;
  double distance_;
};

class CameraModel {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  CameraModel() : initialized_(false) {}
  virtual ~CameraModel() {}

  void setIntrinsicsFromFocalLength(
      const Eigen::Matrix<FloatingPoint, 2, 1>& resolution, double focal_length,
      double min_distance, double max_distance);
  void setIntrinsicsFromFoV(double horizontal_fov, double vertical_fov,
                            double min_distance, double max_distance);
  void setExtrinsics(const Transformation& T_C_B);

  Transformation getCameraPose() const;
  Transformation getBodyPose() const;
  void setCameraPose(const Transformation& cam_pose);
  void setBodyPose(const Transformation& body_pose);

  void getAabb(Point* aabb_min, Point* aabb_max) const;
  bool isPointInView(const Point& point) const;

  const AlignedVector<Plane>& getBoundingPlanes() const {
    return bounding_planes_;
  }
  void getBoundingLines(AlignedVector<Point>* lines) const;

  void getFarPlanePoints(AlignedVector<Point>* points) const;

 private:
  void calculateBoundingPlanes();

  bool initialized_;

  Transformation T_C_B_;
  Transformation T_G_C_;

  AlignedVector<Point> corners_C_;

  AlignedVector<Plane> bounding_planes_;
  Point aabb_min_;
  Point aabb_max_;
};
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_CAMERA_MODEL_H_
```

Includes

- Eigen/Core
- glog/logging.h
- kindr/minimal/quat-transformation.h
- vector
- voxblox/core/common.h (*File common.h*)

Namespaces

- *Namespace voxblox*

Classes

- *Class CameraModel*
- *Class Plane*

File color.h

Contents

- *Definition (voxblox/include/voxblox/core/color.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/core/color.h)

Program Listing for File color.h

Return to documentation for file (voxblox/include/voxblox/core/color.h)

```
#ifndef VOXBLOX_CORE_COLOR_H_
#define VOXBLOX_CORE_COLOR_H_

#include "voxblox/core/common.h"

namespace voxblox {

// Color maps.

inline Color rainbowColorMap(double h) {
  Color color;
  color.a = 255;
```

(continues on next page)

(continued from previous page)

```
  // blend over HSV-values (more colors)

  double s = 1.0;
  double v = 1.0;

  h -= floor(h);
  h *= 6;
  int i;
  double m, n, f;

  i = floor(h);
  f = h - i;
  if (!(i & 1)) f = 1 - f;  // if i is even
  m = v * (1 - s);
  n = v * (1 - s * f);

  switch (i) {
    case 6:
    case 0:
      color.r = 255 * v;
      color.g = 255 * n;
      color.b = 255 * m;
      break;
    case 1:
      color.r = 255 * n;
      color.g = 255 * v;
      color.b = 255 * m;
      break;
    case 2:
      color.r = 255 * m;
      color.g = 255 * v;
      color.b = 255 * n;
      break;
    case 3:
      color.r = 255 * m;
      color.g = 255 * n;
      color.b = 255 * v;
      break;
    case 4:
      color.r = 255 * n;
      color.g = 255 * m;
      color.b = 255 * v;
      break;
    case 5:
      color.r = 255 * v;
      color.g = 255 * m;
      color.b = 255 * n;
      break;
    default:
      color.r = 255;
      color.g = 127;
      color.b = 127;
      break;
  }

  return color;
}
```

(continues on next page)

(continued from previous page)

```
inline Color grayColorMap(double h) {
  Color color;
  color.a = 255;

  color.r = round(h * 255);
  color.b = color.r;
  color.g = color.r;

  return color;
}

inline Color randomColor() {
  Color color;

  color.a = 255;

  color.r = rand() % 256;
  color.b = rand() % 256;
  color.g = rand() % 256;

  return color;
}

}  // namespace voxblox

#endif  // VOXBLOX_CORE_COLOR_H_
```

Includes

- `voxblox/core/common.h` (*File common.h*)

Included By

- *File voxel.h*
- *File color_maps.h*
- *File voxel_utils.h*

Namespaces

- *Namespace voxblox*

File color_maps.h

Contents

- *Definition (voxblox/include/voxblox/utils/color_maps.h)*
- *Includes*

- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/utils/color_maps.h)

Program Listing for File color_maps.h

Return to documentation for file (voxblox/include/voxblox/utils/color_maps.h)

```
#ifndef VOXBLOX_UTILS_COLOR_MAPS_H_
#define VOXBLOX_UTILS_COLOR_MAPS_H_

#include <algorithm>
#include <vector>

#include "voxblox/core/common.h"
#include "voxblox/core/color.h"

namespace voxblox {

class ColorMap {
 public:
  ColorMap() : min_value_(0.0), max_value_(1.0) {}
  virtual ~ColorMap() {}

  void setMinValue(float min_value) { min_value_ = min_value; }

  void setMaxValue(float max_value) { max_value_ = max_value; }

  virtual Color colorLookup(float value) const = 0;

 protected:
  float min_value_;
  float max_value_;
};

class GrayscaleColorMap : public ColorMap {
 public:
  virtual Color colorLookup(float value) const {
    float new_value = std::min(max_value_, std::max(min_value_, value));
    new_value = (new_value - min_value_) / (max_value_ - min_value_);
    return grayColorMap(new_value);
  }
};

class InverseGrayscaleColorMap : public ColorMap {
 public:
  virtual Color colorLookup(float value) const {
    float new_value = std::min(max_value_, std::max(min_value_, value));
    new_value = (new_value - min_value_) / (max_value_ - min_value_);
    return grayColorMap(1.0 - new_value);
  }
};
```

(continues on next page)

(continued from previous page)

```
class RainbowColorMap : public ColorMap {
 public:
  virtual Color colorLookup(float value) const {
    float new_value = std::min(max_value_, std::max(min_value_, value));
    new_value = (new_value - min_value_) / (max_value_ - min_value_);
    return rainbowColorMap(new_value);
  }
};

class InverseRainbowColorMap : public ColorMap {
 public:
  virtual Color colorLookup(float value) const {
    float new_value = std::min(max_value_, std::max(min_value_, value));
    new_value = (new_value - min_value_) / (max_value_ - min_value_);
    return rainbowColorMap(1.0 - new_value);
  }
};

class IronbowColorMap : public ColorMap {
 public:
  IronbowColorMap() : ColorMap() {
    palette_colors_.push_back(Color(0, 0, 0));
    palette_colors_.push_back(Color(145, 20, 145));
    palette_colors_.push_back(Color(255, 138, 0));
    palette_colors_.push_back(Color(255, 230, 40));
    palette_colors_.push_back(Color(255, 255, 255));
    // Add an extra to avoid overflow.
    palette_colors_.push_back(Color(255, 255, 255));

    increment_ = 1.0 / (palette_colors_.size() - 2);
  }

  virtual Color colorLookup(float value) const {
    float new_value = std::min(max_value_, std::max(min_value_, value));
    new_value = (new_value - min_value_) / (max_value_ - min_value_);

    size_t index = static_cast<size_t>(std::floor(new_value / increment_));

    return Color::blendTwoColors(
        palette_colors_[index], increment_ * (index + 1) - new_value,
        palette_colors_[index + 1], new_value - increment_ * (index));
  }

 protected:
  std::vector<Color> palette_colors_;
  float increment_;
};

}  // namespace voxblox

#endif  // VOXBLOX_UTILS_COLOR_MAPS_H_
```

Includes

- algorithm

- vector
- voxblox/core/color.h (*File color.h*)
- voxblox/core/common.h (*File common.h*)

Included By

- *File intensity_server.h*
- *File intensity_vis.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class ColorMap*
- *Class GrayscaleColorMap*
- *Class InverseGrayscaleColorMap*
- *Class InverseRainbowColorMap*
- *Class IronbowColorMap*
- *Class RainbowColorMap*

File common.h

Contents

- *Definition (voxblox/include/voxblox/core/common.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/core/common.h)

Program Listing for File common.h

Return to documentation for file (voxblox/include/voxblox/core/common.h)

```
#ifndef VOXBLOX_CORE_COMMON_H_
#define VOXBLOX_CORE_COMMON_H_

#include <deque>
#include <list>
#include <memory>
#include <queue>
#include <set>
#include <stack>
#include <unordered_map>
#include <unordered_set>
#include <utility>
#include <vector>

#include <glog/logging.h>
#include <kindr/minimal/quat-transformation.h>
#include <Eigen/Core>

namespace voxblox {

// Aligned Eigen containers
template <typename Type>
using AlignedVector = std::vector<Type, Eigen::aligned_allocator<Type>>;
template <typename Type>
using AlignedDeque = std::deque<Type, Eigen::aligned_allocator<Type>>;
template <typename Type>
using AlignedQueue = std::queue<Type, AlignedDeque<Type>>;
template <typename Type>
using AlignedStack = std::stack<Type, AlignedDeque<Type>>;
template <typename Type>
using AlignedList = std::list<Type, Eigen::aligned_allocator<Type>>;

template <typename Type, typename... Arguments>
inline std::shared_ptr<Type> aligned_shared(Arguments&&... arguments) {
  typedef typename std::remove_const<Type>::type TypeNonConst;
  return std::allocate_shared<Type>(Eigen::aligned_allocator<TypeNonConst>(),
                                    std::forward<Arguments>(arguments)...);
}

// Types.
typedef float FloatingPoint;
typedef int IndexElement;
typedef int64_t LongIndexElement;

typedef Eigen::Matrix<FloatingPoint, 3, 1> Point;
typedef Eigen::Matrix<FloatingPoint, 3, 1> Ray;

typedef Eigen::Matrix<IndexElement, 3, 1> AnyIndex;
typedef AnyIndex VoxelIndex;
typedef AnyIndex BlockIndex;
typedef AnyIndex SignedIndex;

typedef Eigen::Matrix<LongIndexElement, 3, 1> LongIndex;
typedef LongIndex GlobalIndex;

typedef std::pair<BlockIndex, VoxelIndex> VoxelKey;
```

(continues on next page)

(continued from previous page)

```
typedef AlignedVector<AnyIndex> IndexVector;
typedef IndexVector BlockIndexList;
typedef IndexVector VoxelIndexList;
typedef AlignedVector<LongIndex> LongIndexVector;
typedef LongIndexVector GlobalIndexVector;

struct Color;
typedef uint32_t Label;
typedef uint32_t LabelConfidence;

// Pointcloud types for external interface.
typedef AlignedVector<Point> Pointcloud;
typedef AlignedVector<Color> Colors;
typedef AlignedVector<Label> Labels;

// For triangle meshing/vertex access.
typedef size_t VertexIndex;
typedef AlignedVector<VertexIndex> VertexIndexList;
typedef Eigen::Matrix<FloatingPoint, 3, 3> Triangle;
typedef AlignedVector<Triangle> TriangleVector;

// Transformation type for defining sensor orientation.
typedef kindr::minimal::QuatTransformationTemplate<FloatingPoint>
    Transformation;
typedef kindr::minimal::RotationQuaternionTemplate<FloatingPoint> Rotation;
typedef kindr::minimal::RotationQuaternionTemplate<
    FloatingPoint>::Implementation Quaternion;

// For alignment of layers / point clouds
typedef Eigen::Matrix<FloatingPoint, 3, Eigen::Dynamic> PointsMatrix;
template <size_t size>
using SquareMatrix = Eigen::Matrix<FloatingPoint, size, size>;

// Interpolation structure
typedef Eigen::Matrix<FloatingPoint, 8, 8> InterpTable;
typedef Eigen::Matrix<FloatingPoint, 1, 8> InterpVector;
// Type must allow negatives:
typedef Eigen::Array<IndexElement, 3, 8> InterpIndexes;

struct Color {
  Color() : r(0), g(0), b(0), a(0) {}
  Color(uint8_t _r, uint8_t _g, uint8_t _b) : Color(_r, _g, _b, 255) {}
  Color(uint8_t _r, uint8_t _g, uint8_t _b, uint8_t _a)
      : r(_r), g(_g), b(_b), a(_a) {}

  uint8_t r;
  uint8_t g;
  uint8_t b;
  uint8_t a;

  static Color blendTwoColors(const Color& first_color,
                              FloatingPoint first_weight,
                              const Color& second_color,
                              FloatingPoint second_weight) {
    FloatingPoint total_weight = first_weight + second_weight;

    first_weight /= total_weight;
```

(continues on next page)

(continued from previous page)

```
    second_weight /= total_weight;

    Color new_color;
    new_color.r = static_cast<uint8_t>(
        round(first_color.r * first_weight + second_color.r * second_weight));
    new_color.g = static_cast<uint8_t>(
        round(first_color.g * first_weight + second_color.g * second_weight));
    new_color.b = static_cast<uint8_t>(
        round(first_color.b * first_weight + second_color.b * second_weight));
    new_color.a = static_cast<uint8_t>(
        round(first_color.a * first_weight + second_color.a * second_weight));

    return new_color;
  }

  // Now a bunch of static colors to use! :)
  static const Color White() { return Color(255, 255, 255); }
  static const Color Black() { return Color(0, 0, 0); }
  static const Color Gray() { return Color(127, 127, 127); }
  static const Color Red() { return Color(255, 0, 0); }
  static const Color Green() { return Color(0, 255, 0); }
  static const Color Blue() { return Color(0, 0, 255); }
  static const Color Yellow() { return Color(255, 255, 0); }
  static const Color Orange() { return Color(255, 127, 0); }
  static const Color Purple() { return Color(127, 0, 255); }
  static const Color Teal() { return Color(0, 255, 255); }
  static const Color Pink() { return Color(255, 0, 127); }
};

// Constants used across the library.
constexpr FloatingPoint kEpsilon = 1e-6;
constexpr float kFloatEpsilon = 1e-6;
// Grid <-> point conversion functions.

template <typename IndexType>
inline IndexType getGridIndexFromPoint(const Point& point,
                                       const FloatingPoint grid_size_inv) {
  return IndexType(std::floor(point.x() * grid_size_inv + kEpsilon),
                   std::floor(point.y() * grid_size_inv + kEpsilon),
                   std::floor(point.z() * grid_size_inv + kEpsilon));
}

template <typename IndexType>
inline IndexType getGridIndexFromPoint(const Point& scaled_point) {
  return IndexType(std::floor(scaled_point.x() + kEpsilon),
                   std::floor(scaled_point.y() + kEpsilon),
                   std::floor(scaled_point.z() + kEpsilon));
}

template <typename IndexType>
inline IndexType getGridIndexFromOriginPoint(
    const Point& point, const FloatingPoint grid_size_inv) {
  return IndexType(std::round(point.x() * grid_size_inv),
                   std::round(point.y() * grid_size_inv),
                   std::round(point.z() * grid_size_inv));
}
```

(continues on next page)

(continued from previous page)

```
template <typename IndexType>
inline Point getCenterPointFromGridIndex(const IndexType& idx,
                                         FloatingPoint grid_size) {
  return Point((static_cast<FloatingPoint>(idx.x()) + 0.5) * grid_size,
               (static_cast<FloatingPoint>(idx.y()) + 0.5) * grid_size,
               (static_cast<FloatingPoint>(idx.z()) + 0.5) * grid_size);
}

template <typename IndexType>
inline Point getOriginPointFromGridIndex(const IndexType& idx,
                                         FloatingPoint grid_size) {
  return Point(static_cast<FloatingPoint>(idx.x()) * grid_size,
               static_cast<FloatingPoint>(idx.y()) * grid_size,
               static_cast<FloatingPoint>(idx.z()) * grid_size);
}

inline GlobalIndex getGlobalVoxelIndexFromBlockAndVoxelIndex(
    const BlockIndex& block_index, const VoxelIndex& voxel_index,
    int voxels_per_side) {
  return GlobalIndex(block_index.cast<LongIndexElement>() * voxels_per_side +
                     voxel_index.cast<LongIndexElement>());
}

inline BlockIndex getBlockIndexFromGlobalVoxelIndex(
    const GlobalIndex& global_voxel_idx, FloatingPoint voxels_per_side_inv) {
  return BlockIndex(
      std::floor(static_cast<FloatingPoint>(global_voxel_idx.x()) *
                 voxels_per_side_inv),
      std::floor(static_cast<FloatingPoint>(global_voxel_idx.y()) *
                 voxels_per_side_inv),
      std::floor(static_cast<FloatingPoint>(global_voxel_idx.z()) *
                 voxels_per_side_inv));
}

inline bool isPowerOfTwo(int x) { return (x & (x - 1)) == 0; }

inline VoxelIndex getLocalFromGlobalVoxelIndex(
    const GlobalIndex& global_voxel_idx, const int voxels_per_side) {
  // add a big number to the index to make it positive
  constexpr int offset = 1 << (8 * sizeof(IndexElement) - 1);

  CHECK(isPowerOfTwo(voxels_per_side));

  return VoxelIndex((global_voxel_idx.x() + offset) & (voxels_per_side - 1),
                    (global_voxel_idx.y() + offset) & (voxels_per_side - 1),
                    (global_voxel_idx.z() + offset) & (voxels_per_side - 1));
}

inline void getBlockAndVoxelIndexFromGlobalVoxelIndex(
    const GlobalIndex& global_voxel_idx, const int voxels_per_side,
    BlockIndex* block_index, VoxelIndex* voxel_index) {
  CHECK_NOTNULL(block_index);
  CHECK_NOTNULL(voxel_index);
  const FloatingPoint voxels_per_side_inv = 1.0 / voxels_per_side;
  *block_index =
      getBlockIndexFromGlobalVoxelIndex(global_voxel_idx, voxels_per_side_inv);
  *voxel_index =
```

(continues on next page)

(continued from previous page)

```
      getLocalFromGlobalVoxelIndex(global_voxel_idx, voxels_per_side);
}

// Math functions.
inline int signum(FloatingPoint x) { return (x == 0) ? 0 : x < 0 ? -1 : 1; }

// For occupancy/octomap-style mapping.
inline float logOddsFromProbability(float probability) {
  CHECK(probability >= 0.0f && probability <= 1.0f);
  return log(probability / (1.0 - probability));
}

inline float probabilityFromLogOdds(float log_odds) {
  return 1.0 - (1.0 / (1.0 + exp(log_odds)));
}

inline void transformPointcloud(const Transformation& T_N_O,
                                const Pointcloud& ptcloud,
                                Pointcloud* ptcloud_out) {
  ptcloud_out->clear();
  ptcloud_out->resize(ptcloud.size());

  for (size_t i = 0; i < ptcloud.size(); ++i) {
    (*ptcloud_out)[i] = T_N_O * ptcloud[i];
  }
}

}  // namespace voxblox

#endif  // VOXBLOX_CORE_COMMON_H_
```

Includes

- Eigen/Core
- deque
- glog/logging.h
- kindr/minimal/quat-transformation.h
- list
- memory
- queue (*File bucket_queue.h*)
- set
- stack
- unordered_map
- unordered_set
- utility
- vector

Included By

- *File icp.h*
- *File block.h*
- *File block_hash.h*
- *File color.h*
- *File esdf_map.h*
- *File layer.h*
- *File occupancy_map.h*
- *File tsdf_map.h*
- *File voxel.h*
- *File integrator_utils.h*
- *File merge_integration.h*
- *File tsdf_integrator.h*
- *File interpolator.h*
- *File layer_io.h*
- *File ply_writer.h*
- *File mesh.h*
- *File mesh_layer.h*
- *File mesh_utils.h*
- *File objects.h*
- *File simulation_world.h*
- *File approx_hash_array.h*
- *File bucket_queue.h*
- *File camera_model.h*
- *File color_maps.h*
- *File distance_utils.h*
- *File layer_utils.h*
- *File meshing_utils.h*
- *File neighbor_tools.h*
- *File timing.h*
- *File voxel_utils.h*
- *File conversions.h*
- *File interactive_slider.h*
- *File mesh_pcl.h*
- *File mesh_vis.h*
- *File ptcloud_vis.h*

- *File transformer.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct Color*

File conversions.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/conversions.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/conversions.h)

Program Listing for File conversions.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/conversions.h)

```
#ifndef VOXBLOX_ROS_CONVERSIONS_H_
#define VOXBLOX_ROS_CONVERSIONS_H_

#include <pcl/point_types.h>
#include <pcl_ros/point_cloud.h>
#include <std_msgs/ColorRGBA.h>
#include <algorithm>
#include <vector>

#include <voxblox/core/common.h>
#include <voxblox/core/layer.h>
#include <voxblox/mesh/mesh.h>
#include <voxblox_msgs/Layer.h>

namespace voxblox {

enum class MapDerializationAction : uint8_t {
  kUpdate = 0u,
  kMerge = 1u,
  kReset = 2u
};

inline void colorVoxbloxToMsg(const Color& color,
```

(continues on next page)

(continued from previous page)

```
                                std_msgs::ColorRGBA* color_msg) {
  CHECK_NOTNULL(color_msg);
  color_msg->r = color.r / 255.0;
  color_msg->g = color.g / 255.0;
  color_msg->b = color.b / 255.0;
  color_msg->a = color.a / 255.0;
}

inline void colorMsgToVoxblox(const std_msgs::ColorRGBA& color_msg,
                              Color* color) {
  CHECK_NOTNULL(color);
  color->r = static_cast<uint8_t>(color_msg.r * 255.0);
  color->g = static_cast<uint8_t>(color_msg.g * 255.0);
  color->b = static_cast<uint8_t>(color_msg.b * 255.0);
  color->a = static_cast<uint8_t>(color_msg.a * 255.0);
}

inline void pointcloudToPclXYZRGB(
    const Pointcloud& ptcloud, const Colors& colors,
    pcl::PointCloud<pcl::PointXYZRGB>* ptcloud_pcl) {
  CHECK_NOTNULL(ptcloud_pcl);
  ptcloud_pcl->clear();
  ptcloud_pcl->reserve(ptcloud.size());
  for (size_t i = 0; i < ptcloud.size(); ++i) {
    pcl::PointXYZRGB point;
    point.x = ptcloud[i].x();
    point.y = ptcloud[i].y();
    point.z = ptcloud[i].z();

    point.r = colors[i].r;
    point.g = colors[i].g;
    point.b = colors[i].b;

    ptcloud_pcl->push_back(point);
  }
}

inline void pointcloudToPclXYZ(const Pointcloud& ptcloud,
                               pcl::PointCloud<pcl::PointXYZ>* ptcloud_pcl) {
  CHECK_NOTNULL(ptcloud_pcl);
  ptcloud_pcl->clear();
  ptcloud_pcl->reserve(ptcloud.size());
  for (size_t i = 0; i < ptcloud.size(); ++i) {
    pcl::PointXYZ point;
    point.x = ptcloud[i].x();
    point.y = ptcloud[i].y();
    point.z = ptcloud[i].z();

    ptcloud_pcl->push_back(point);
  }
}

inline void pointcloudToPclXYZI(const Pointcloud& ptcloud,
                                const std::vector<float>& intensities,
                                pcl::PointCloud<pcl::PointXYZI>* ptcloud_pcl) {
  CHECK_NOTNULL(ptcloud_pcl);
  CHECK_EQ(ptcloud.size(), intensities.size());
```

(continues on next page)

(continued from previous page)

```
  ptcloud_pcl->clear();
  ptcloud_pcl->reserve(ptcloud.size());
  for (size_t i = 0; i < ptcloud.size(); ++i) {
    pcl::PointXYZI point;
    point.x = ptcloud[i].x();
    point.y = ptcloud[i].y();
    point.z = ptcloud[i].z();
    point.intensity = intensities[i];

    ptcloud_pcl->push_back(point);
  }
}

// Declarations
template <typename VoxelType>
void serializeLayerAsMsg(
    const Layer<VoxelType>& layer, const bool only_updated,
    voxblox_msgs::Layer* msg,
    const MapDerializationAction& action = MapDerializationAction::kUpdate);

template <typename VoxelType>
bool deserializeMsgToLayer(const voxblox_msgs::Layer& msg,
                           Layer<VoxelType>* layer);

template <typename VoxelType>
bool deserializeMsgToLayer(const voxblox_msgs::Layer& msg,
                           const MapDerializationAction& action,
                           Layer<VoxelType>* layer);

}  // namespace voxblox

#endif  // VOXBLOX_ROS_CONVERSIONS_H_

#include "voxblox_ros/conversions_inl.h"
```

Includes

- algorithm
- pcl/point_types.h
- pcl_ros/point_cloud.h
- std_msgs/ColorRGBA.h
- vector
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/mesh/mesh.h (*File mesh.h*)
- voxblox_msgs/Layer.h
- voxblox_ros/conversions_inl.h (*File conversions_inl.h*)

Included By

- *File intensity_vis.h*
- *File mesh_vis.h*
- *File ptcloud_vis.h*
- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

File conversions_inl.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/conversions_inl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/conversions_inl.h)

Program Listing for File conversions_inl.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/conversions_inl.h)

```
#ifndef VOXBLOX_ROS_CONVERSIONS_INL_H_
#define VOXBLOX_ROS_CONVERSIONS_INL_H_

#include <vector>

namespace voxblox {

template <typename VoxelType>
void serializeLayerAsMsg(const Layer<VoxelType>& layer, const bool only_updated,
                         voxblox_msgs::Layer* msg,
                         const MapDerializationAction& action) {
  CHECK_NOTNULL(msg);
  msg->voxels_per_side = layer.voxels_per_side();
  msg->voxel_size = layer.voxel_size();

  msg->layer_type = getVoxelType<VoxelType>();

  BlockIndexList block_list;
  if (only_updated) {
    layer.getAllUpdatedBlocks(&block_list);
  } else {
```

(continues on next page)

(continued from previous page)

```
    layer.getAllAllocatedBlocks(&block_list);
  }

  msg->action = static_cast<uint8_t>(action);

  voxblox_msgs::Block block_msg;
  msg->blocks.reserve(block_list.size());
  for (const BlockIndex& index : block_list) {
    block_msg.x_index = index.x();
    block_msg.y_index = index.y();
    block_msg.z_index = index.z();

    std::vector<uint32_t> data;
    layer.getBlockByIndex(index).serializeToIntegers(&data);

    block_msg.data = data;
    msg->blocks.push_back(block_msg);
  }
}  // namespace voxblox

template <typename VoxelType>
bool deserializeMsgToLayer(const voxblox_msgs::Layer& msg,
                           Layer<VoxelType>* layer) {
  CHECK_NOTNULL(layer);
  return deserializeMsgToLayer<VoxelType>(
      msg, static_cast<MapDerializationAction>(msg.action), layer);
}

template <typename VoxelType>
bool deserializeMsgToLayer(const voxblox_msgs::Layer& msg,
                           const MapDerializationAction& action,
                           Layer<VoxelType>* layer) {
  CHECK_NOTNULL(layer);
  if (getVoxelType<VoxelType>().compare(msg.layer_type) != 0) {
    return false;
  }

  // So we also need to check if the sizes match. If they don't, we can't
  // parse this at all.
  constexpr double kVoxelSizeEpsilon = 1e-5;
  if (msg.voxels_per_side != layer->voxels_per_side() ||
      std::abs(msg.voxel_size - layer->voxel_size()) > kVoxelSizeEpsilon) {
    LOG(ERROR) << "Sizes don't match!";
    return false;
  }

  if (action == MapDerializationAction::kReset) {
    layer->removeAllBlocks();
  }

  for (const voxblox_msgs::Block& block_msg : msg.blocks) {
    BlockIndex index(block_msg.x_index, block_msg.y_index, block_msg.z_index);

    // Either we want to update an existing block or there was no block there
    // before.
    if (action == MapDerializationAction::kUpdate || !layer->hasBlock(index)) {
      // Create a new block if it doesn't exist yet, or get the existing one
```

(continues on next page)

(continued from previous page)

```
      // at the correct block index.
      typename Block<VoxelType>::Ptr block_ptr =
          layer->allocateBlockPtrByIndex(index);

      std::vector<uint32_t> data = block_msg.data;
      block_ptr->deserializeFromIntegers(data);

    } else if (action == MapDerializationAction::kMerge) {
      typename Block<VoxelType>::Ptr old_block_ptr =
          layer->getBlockPtrByIndex(index);
      CHECK(old_block_ptr);

      typename Block<VoxelType>::Ptr new_block_ptr(new Block<VoxelType>(
          old_block_ptr->voxels_per_side(), old_block_ptr->voxel_size(),
          old_block_ptr->origin()));

      std::vector<uint32_t> data = block_msg.data;
      new_block_ptr->deserializeFromIntegers(data);

      old_block_ptr->mergeBlock(*new_block_ptr);
    }
  }

  switch (action) {
    case MapDerializationAction::kReset:
      CHECK_EQ(layer->getNumberOfAllocatedBlocks(), msg.blocks.size());
      break;
    case MapDerializationAction::kUpdate:
      // Fall through intended.
    case MapDerializationAction::kMerge:
      CHECK_GE(layer->getNumberOfAllocatedBlocks(), msg.blocks.size());
      break;
  }

  return true;
}

}  // namespace voxblox

#endif  // VOXBLOX_ROS_CONVERSIONS_INL_H_
```

Includes

- vector

Included By

- *File conversions.h*

Namespaces

- *Namespace voxblox*

File distance_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/distance_utils.h)*
- *Includes*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/distance_utils.h)

Program Listing for File distance_utils.h

Return to documentation for file (voxblox/include/voxblox/utils/distance_utils.h)

```
#ifndef VOXBLOX_UTILS_DISTANCE_UTILS_H_
#define VOXBLOX_UTILS_DISTANCE_UTILS_H_

#include <voxblox/core/block.h>
#include <voxblox/core/common.h>
#include <voxblox/core/layer.h>
#include <voxblox/core/voxel.h>

namespace voxblox {

template <typename VoxelType>
bool getSurfaceDistanceAlongRay(const Layer<VoxelType>& layer,
                                const Point& ray_origin,
                                const Point& bearing_vector,
                                FloatingPoint max_distance,
                                Point* triangulated_pose) {
  CHECK_NOTNULL(triangulated_pose);
  // Make sure bearing vector is normalized.
  const Point ray_direction = bearing_vector.normalized();

  // Keep track of current distance along the ray.
  FloatingPoint t = 0.0;
  // General ray equations: p = o + d * t

  // Cache voxel sizes for faster moving.
  const FloatingPoint voxel_size = layer.voxel_size();

  bool surface_found = false;

  while (t < max_distance) {
    const Point current_pos = ray_origin + t * ray_direction;
    typename Block<VoxelType>::ConstPtr block_ptr =
        layer.getBlockPtrByCoordinates(current_pos);
    if (!block_ptr) {
      // How much should we move up by? 1 voxel? 1 block? Could be close to the
      // block boundary though....
      // Let's start with the naive choice: 1 voxel.
      t += voxel_size;
```

(continues on next page)

(continued from previous page)

```
      continue;
    }
    const VoxelType& voxel = block_ptr->getVoxelByCoordinates(current_pos);
    if (voxel.weight < 1e-6) {
      t += voxel_size;
      continue;
    }
    if (voxel.distance > voxel_size) {
      // Move forward as much as we can.
      t += voxel.distance;
      continue;
    }
    // The other cases are when we are actually at or behind the surface.
    // TODO(helenol): these are gross generalizations; use actual interpolation
    // to get the surface boundary.
    if (voxel.distance < 0.0) {
      surface_found = true;
      break;
    }
    if (voxel.distance < voxel_size) {
      // Also assume this is finding the surface.
      surface_found = true;
      t += voxel.distance;
      break;
    }
    // Default case...
    t += voxel_size;
  }

  if (surface_found) {
    *triangulated_pose = ray_origin + t * ray_direction;
  }

  return surface_found;
}

}  // namespace voxblox

#endif  // VOXBLOX_UTILS_DISTANCE_UTILS_H_
```

Includes

- `voxblox/core/block.h` (*File block.h*)
- `voxblox/core/common.h` (*File common.h*)
- `voxblox/core/layer.h` (*File layer.h*)
- `voxblox/core/voxel.h` (*File voxel.h*)

Namespaces

- *Namespace voxblox*

File esdf_integrator.h

Contents

- *Definition (voxblox/include/voxblox/integrator/esdf_integrator.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/integrator/esdf_integrator.h)

Program Listing for File esdf_integrator.h

Return to documentation for file (voxblox/include/voxblox/integrator/esdf_integrator.h)

```
#ifndef VOXBLOX_INTEGRATOR_ESDF_INTEGRATOR_H_
#define VOXBLOX_INTEGRATOR_ESDF_INTEGRATOR_H_

#include <glog/logging.h>
#include <Eigen/Core>
#include <algorithm>
#include <queue>
#include <utility>
#include <vector>

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/utils/bucket_queue.h"
#include "voxblox/utils/neighbor_tools.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

class EsdfIntegrator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    bool full_euclidean_distance = false;
    FloatingPoint max_distance_m = 2.0;
    FloatingPoint min_distance_m = 0.2;
    FloatingPoint default_distance_m = 2.0;
    FloatingPoint min_diff_m = 0.001;
    float min_weight = 1e-6;
    int num_buckets = 20;
    bool multi_queue = false;
    bool add_occupied_crust = false;
```

(continues on next page)

(continued from previous page)

```
  FloatingPoint clear_sphere_radius = 1.5;
  FloatingPoint occupied_sphere_radius = 5.0;
};

EsdfIntegrator(const Config& config, Layer<TsdfVoxel>* tsdf_layer,
               Layer<EsdfVoxel>* esdf_layer);

void addNewRobotPosition(const Point& position);

void updateFromTsdfLayerBatch();
void updateFromTsdfLayer(bool clear_updated_flag);

void updateFromTsdfBlocks(const BlockIndexList& tsdf_blocks,
                          bool incremental = false);

void processRaiseSet();

void processOpenSet();

bool updateVoxelFromNeighbors(const GlobalIndex& global_index);

// Convenience functions.
inline bool isFixed(FloatingPoint dist_m) const {
  return std::abs(dist_m) < config_.min_distance_m;
}
void clear() {
  updated_blocks_.clear();
  open_.clear();
  raise_ = AlignedQueue<GlobalIndex>();
}
float getEsdfMaxDistance() const { return config_.max_distance_m; }
void setEsdfMaxDistance(float max_distance) {
  config_.max_distance_m = max_distance;
  if (config_.default_distance_m < max_distance) {
    config_.default_distance_m = max_distance;
  }
}
bool getFullEuclidean() const { return config_.full_euclidean_distance; }
void setFullEuclidean(bool full_euclidean) {
  config_.full_euclidean_distance = full_euclidean;
}

protected:
Config config_;

Layer<TsdfVoxel>* tsdf_layer_;
Layer<EsdfVoxel>* esdf_layer_;

BucketQueue<GlobalIndex> open_;

AlignedQueue<GlobalIndex> raise_;

size_t voxels_per_side_;
FloatingPoint voxel_size_;

IndexSet updated_blocks_;
```

(continues on next page)

(continued from previous page)

```
};

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_ESDF_INTEGRATOR_H_
```

Includes

- Eigen/Core
- algorithm
- glog/logging.h
- queue (*File bucket_queue.h*)
- utility
- vector
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/integrator_utils.h (*File integrator_utils.h*)
- voxblox/utils/bucket_queue.h (*File bucket_queue.h*)
- voxblox/utils/neighbor_tools.h (*File neighbor_tools.h*)
- voxblox/utils/timing.h (*File timing.h*)

Included By

- *File esdf_server.h*
- *File mesh_vis.h*
- *File ros_params.h*
- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct EsdfIntegrator::Config*
- *Class EsdfIntegrator*

File esdf_map.h

Contents

- *Definition (voxblox/include/voxblox/core/esdf_map.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/core/esdf_map.h)

Program Listing for File esdf_map.h

Return to documentation for file (voxblox/include/voxblox/core/esdf_map.h)

```
#ifndef VOXBLOX_CORE_ESDF_MAP_H_
#define VOXBLOX_CORE_ESDF_MAP_H_

#include <glog/logging.h>
#include <memory>
#include <string>
#include <utility>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/interpolator/interpolator.h"

#include "voxblox/io/layer_io.h"

namespace voxblox {
class EsdfMap {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<EsdfMap> Ptr;

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    FloatingPoint esdf_voxel_size = 0.2;
    size_t esdf_voxels_per_side = 16u;
  };

  explicit EsdfMap(const Config& config)
      : esdf_layer_(new Layer<EsdfVoxel>(config.esdf_voxel_size,
                                         config.esdf_voxels_per_side)),
        interpolator_(esdf_layer_.get()) {
    block_size_ = config.esdf_voxel_size * config.esdf_voxels_per_side;
  }
```

(continues on next page)

(continued from previous page)

```
explicit EsdfMap(const Layer<EsdfVoxel>& layer)
    : EsdfMap(aligned_shared<Layer<EsdfVoxel>>(layer)) {}

explicit EsdfMap(Layer<EsdfVoxel>::Ptr layer)
    : esdf_layer_(layer), interpolator_(CHECK_NOTNULL(esdf_layer_.get())) {
  block_size_ = layer->block_size();
}

virtual ~EsdfMap() {}

Layer<EsdfVoxel>* getEsdfLayerPtr() { return esdf_layer_.get(); }
const Layer<EsdfVoxel>& getEsdfLayer() const { return *esdf_layer_; }

FloatingPoint block_size() const { return block_size_; }
FloatingPoint voxel_size() const { return esdf_layer_->voxel_size(); }

bool getDistanceAtPosition(const Eigen::Vector3d& position,
                           double* distance) const;
bool getDistanceAtPosition(const Eigen::Vector3d& position, bool interpolate,
                           double* distance) const;

bool getDistanceAndGradientAtPosition(const Eigen::Vector3d& position,
                                      double* distance,
                                      Eigen::Vector3d* gradient) const;
bool getDistanceAndGradientAtPosition(const Eigen::Vector3d& position,
                                      bool interpolate, double* distance,
                                      Eigen::Vector3d* gradient) const;

bool isObserved(const Eigen::Vector3d& position) const;

// NOTE(mereweth@jpl.nasa.gov)
// EigenDRef is fully dynamic stride type alias for Numpy array slices
// Use column-major matrices; column-by-column traversal is faster
// Convenience alias borrowed from pybind11
using EigenDStride = Eigen::Stride<Eigen::Dynamic, Eigen::Dynamic>;
template <typename MatrixType>
using EigenDRef = Eigen::Ref<MatrixType, 0, EigenDStride>;

// Convenience functions for querying many points at once from Python
void batchGetDistanceAtPosition(
    EigenDRef<const Eigen::Matrix<double, 3, Eigen::Dynamic>>& positions,
    Eigen::Ref<Eigen::VectorXd> distances,
    Eigen::Ref<Eigen::VectorXi> observed) const;

void batchGetDistanceAndGradientAtPosition(
    EigenDRef<const Eigen::Matrix<double, 3, Eigen::Dynamic>>& positions,
    Eigen::Ref<Eigen::VectorXd> distances,
    EigenDRef<Eigen::Matrix<double, 3, Eigen::Dynamic>>& gradients,
    Eigen::Ref<Eigen::VectorXi> observed) const;

void batchIsObserved(
    EigenDRef<const Eigen::Matrix<double, 3, Eigen::Dynamic>>& positions,
    Eigen::Ref<Eigen::VectorXi> observed) const;

unsigned int coordPlaneSliceGetCount(unsigned int free_plane_index,
                                     double free_plane_val) const;
```

(continues on next page)

(continued from previous page)

```
  unsigned int coordPlaneSliceGetDistance(
      unsigned int free_plane_index, double free_plane_val,
      EigenDRef<Eigen::Matrix<double, 3, Eigen::Dynamic>>& positions,
      Eigen::Ref<Eigen::VectorXd> distances, unsigned int max_points) const;

 protected:
  FloatingPoint block_size_;

  // The layers.
  Layer<EsdfVoxel>::Ptr esdf_layer_;

  // Interpolator for the layer.
  Interpolator<EsdfVoxel> interpolator_;
};

}  // namespace voxblox

#endif  // VOXBLOX_CORE_ESDF_MAP_H_
```

Includes

- glog/logging.h
- memory
- string
- utility
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/interpolator/interpolator.h (*File interpolator.h*)
- voxblox/io/layer_io.h (*File layer_io.h*)

Included By

- *File esdf_server.h*
- *File ros_params.h*
- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct EsdfMap::Config*
- *Class EsdfMap*

File esdf_occ_integrator.h

Contents

- *Definition (voxblox/include/voxblox/integrator/esdf_occ_integrator.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/integrator/esdf_occ_integrator.h)

Program Listing for File esdf_occ_integrator.h

Return to documentation for file (voxblox/include/voxblox/integrator/esdf_occ_integrator.h)

```
#ifndef VOXBLOX_INTEGRATOR_ESDF_OCC_INTEGRATOR_H_
#define VOXBLOX_INTEGRATOR_ESDF_OCC_INTEGRATOR_H_

#include <glog/logging.h>
#include <Eigen/Core>
#include <algorithm>
#include <queue>
#include <utility>
#include <vector>

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/utils/bucket_queue.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

class EsdfOccIntegrator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    FloatingPoint max_distance_m = 2.0;
    FloatingPoint default_distance_m = 2.0;
```

(continues on next page)

(continued from previous page)

```
    int num_buckets = 20;
  };

  EsdfOccIntegrator(const Config& config, Layer<OccupancyVoxel>* occ_layer,
                    Layer<EsdfVoxel>* esdf_layer);

  void updateFromOccLayerBatch();
  void updateFromOccBlocks(const BlockIndexList& occ_blocks);

  void processOpenSet();

  void getNeighborsAndDistances(
      const BlockIndex& block_index, const VoxelIndex& voxel_index,
      AlignedVector<VoxelKey>* neighbors, AlignedVector<float>* distances,
      AlignedVector<Eigen::Vector3i>* directions) const;
  void getNeighbor(const BlockIndex& block_index, const VoxelIndex& voxel_index,
                   const Eigen::Vector3i& direction,
                   BlockIndex* neighbor_block_index,
                   VoxelIndex* neighbor_voxel_index) const;

 protected:
  Config config_;

  Layer<OccupancyVoxel>* occ_layer_;
  Layer<EsdfVoxel>* esdf_layer_;

  BucketQueue<VoxelKey> open_;

  AlignedQueue<VoxelKey> raise_;

  size_t esdf_voxels_per_side_;
  FloatingPoint esdf_voxel_size_;
};

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_ESDF_OCC_INTEGRATOR_H_
```

Includes

- Eigen/Core
- algorithm
- glog/logging.h
- queue (*File bucket_queue.h*)
- utility
- vector
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/integrator_utils.h (*File integrator_utils.h*)
- voxblox/utils/bucket_queue.h (*File bucket_queue.h*)

- voxblox/utils/timing.h (*File timing.h*)

Included By

- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct EsdfOccIntegrator::Config*
- *Class EsdfOccIntegrator*

File esdf_server.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/esdf_server.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox_ros/include/voxblox_ros/esdf_server.h)

Program Listing for File esdf_server.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/esdf_server.h)

```
#ifndef VOXBLOX_ROS_ESDF_SERVER_H_
#define VOXBLOX_ROS_ESDF_SERVER_H_

#include <memory>
#include <string>

#include <voxblox/core/esdf_map.h>
#include <voxblox/integrator/esdf_integrator.h>
#include <voxblox_msgs/Layer.h>

#include "voxblox_ros/tsdf_server.h"

namespace voxblox {

class EsdfServer : public TsdfServer {
 public:
```

(continues on next page)

(continued from previous page)

```
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  EsdfServer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private);
  EsdfServer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private,
             const EsdfMap::Config& esdf_config,
             const EsdfIntegrator::Config& esdf_integrator_config,
             const TsdfMap::Config& tsdf_config,
             const TsdfIntegratorBase::Config& tsdf_integrator_config);
  virtual ~EsdfServer() {}

  bool generateEsdfCallback(std_srvs::Empty::Request& request,     // NOLINT
                            std_srvs::Empty::Response& response);  // NOLINT

  void publishAllUpdatedEsdfVoxels();
  virtual void publishSlices();
  void publishTraversable();

  virtual void updateMesh();
  virtual void publishPointclouds();
  virtual void newPoseCallback(const Transformation& T_G_C);
  virtual void publishMap(const bool reset_remote_map = false);
  virtual bool saveMap(const std::string& file_path);
  virtual bool loadMap(const std::string& file_path);

  void updateEsdf();
  // Update the ESDF all at once; clear the existing map.
  void updateEsdfBatch(bool full_euclidean = false);

  // Overwrites the layer with what's coming from the topic!
  void esdfMapCallback(const voxblox_msgs::Layer& layer_msg);

  inline std::shared_ptr<EsdfMap> getEsdfMapPtr() { return esdf_map_; }
  inline std::shared_ptr<const EsdfMap> getEsdfMapPtr() const {
    return esdf_map_;
  }

  bool getClearSphere() const { return clear_sphere_for_planning_; }
  void setClearSphere(bool clear_sphere_for_planning) {
    clear_sphere_for_planning_ = clear_sphere_for_planning;
  }
  float getEsdfMaxDistance() const;
  void setEsdfMaxDistance(float max_distance);
  float getTraversabilityRadius() const;
  void setTraversabilityRadius(float traversability_radius);

  void disableIncrementalUpdate() { incremental_update_ = false; }
  void enableIncrementalUpdate() { incremental_update_ = true; }

  virtual void clear();

 protected:
  void setupRos();

  // Publish markers for visualization.
  ros::Publisher esdf_pointcloud_pub_;
  ros::Publisher esdf_slice_pub_;
  ros::Publisher traversable_pub_;
```

(continues on next page)

(continued from previous page)

```
  ros::Publisher esdf_map_pub_;

  ros::Subscriber esdf_map_sub_;

  ros::ServiceServer generate_esdf_srv_;

  bool clear_sphere_for_planning_;
  bool publish_esdf_map_;
  bool publish_traversable_;
  float traversability_radius_;
  bool incremental_update_;

  // ESDF maps.
  std::shared_ptr<EsdfMap> esdf_map_;
  std::unique_ptr<EsdfIntegrator> esdf_integrator_;
};

}  // namespace voxblox

#endif  // VOXBLOX_ROS_ESDF_SERVER_H_
```

Includes

- memory
- string
- voxblox/core/esdf_map.h (*File esdf_map.h*)
- voxblox/integrator/esdf_integrator.h (*File esdf_integrator.h*)
- voxblox_msgs/Layer.h
- voxblox_ros/tsdf_server.h (*File tsdf_server.h*)

Namespaces

- *Namespace voxblox*

Classes

- *Class EsdfServer*

File evaluation_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/evaluation_utils.h)*
- *Includes*

- *Included By*
- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/utils/evaluation_utils.h`)

Program Listing for File evaluation_utils.h

Return to documentation for file (`voxblox/include/voxblox/utils/evaluation_utils.h`)

```
#ifndef VOXBLOX_UTILS_EVALUATION_UTILS_H_
#define VOXBLOX_UTILS_EVALUATION_UTILS_H_

#include <algorithm>
#include <string>

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
namespace utils {

enum class VoxelEvaluationResult { kNoOverlap, kIgnored, kEvaluated };

enum class VoxelEvaluationMode {
  kEvaluateAllVoxels,
  kIgnoreErrorBehindTestSurface,
  kIgnoreErrorBehindGtSurface,
  kIgnoreErrorBehindAllSurfaces
};

struct VoxelEvaluationDetails {
  FloatingPoint rmse = 0.0;
  // Max and min of absolute distance error.
  FloatingPoint max_error = 0.0;
  FloatingPoint min_error = 0.0;
  size_t num_evaluated_voxels = 0u;
  size_t num_ignored_voxels = 0u;
  size_t num_overlapping_voxels = 0u;
  size_t num_non_overlapping_voxels = 0u;

  std::string toString() const {
    std::stringstream ss;
    ss << "\n\n======= Layer Evaluation Results =======\n"
       << " num evaluated voxels:       " << num_evaluated_voxels << "\n"
       << " num overlapping voxels:     " << num_overlapping_voxels << "\n"
       << " num non-overlapping voxels: " << num_non_overlapping_voxels << "\n"
       << " num ignored voxels:         " << num_ignored_voxels << "\n"
       << " error min:                  " << min_error << "\n"
       << " error max:                  " << max_error << "\n"
       << " RMSE:                       " << rmse << "\n"
       << "========================================\n";
    return ss.str();
  }
```

(continues on next page)

(continued from previous page)

```
};

template <typename VoxelType>
VoxelEvaluationResult computeVoxelError(
    const VoxelType& voxel_gt, const VoxelType& voxel_test,
    const VoxelEvaluationMode evaluation_mode, FloatingPoint* error);

template <typename VoxelType>
bool isObservedVoxel(const VoxelType& voxel);

template <typename VoxelType>
FloatingPoint getVoxelSdf(const VoxelType& voxel);

template <typename VoxelType>
void setVoxelSdf(const FloatingPoint sdf, VoxelType* voxel);

template <typename VoxelType>
void setVoxelWeight(const FloatingPoint weight, VoxelType* voxel);

template <typename VoxelType>
FloatingPoint evaluateLayersRmse(
    const Layer<VoxelType>& layer_gt, const Layer<VoxelType>& layer_test,
    const VoxelEvaluationMode& voxel_evaluation_mode,
    VoxelEvaluationDetails* evaluation_result = nullptr,
    Layer<VoxelType>* error_layer = nullptr) {
  // Iterate over all voxels in the test layer and look them up in the ground
  // truth layer. Then compute RMSE.
  BlockIndexList block_list;
  layer_test.getAllAllocatedBlocks(&block_list);
  size_t vps = layer_test.voxels_per_side();
  size_t num_voxels_per_block = vps * vps * vps;

  VoxelEvaluationDetails evaluation_details;

  double total_squared_error = 0.0;

  for (const BlockIndex& block_index : block_list) {
    const Block<VoxelType>& test_block =
        layer_test.getBlockByIndex(block_index);

    if (!layer_gt.hasBlock(block_index)) {
      for (size_t linear_index = 0u; linear_index < num_voxels_per_block;
           ++linear_index) {
        const VoxelType& voxel = test_block.getVoxelByLinearIndex(linear_index);
        if (isObservedVoxel(voxel)) {
          ++evaluation_details.num_non_overlapping_voxels;
        }
      }
      continue;
    }
    const Block<VoxelType>& gt_block = layer_gt.getBlockByIndex(block_index);

    typename Block<VoxelType>::Ptr error_block;
    if (error_layer != nullptr) {
      error_block = error_layer->allocateBlockPtrByIndex(block_index);
    }
```

(continues on next page)

(continued from previous page)

```
    for (size_t linear_index = 0u; linear_index < num_voxels_per_block;
         ++linear_index) {
      FloatingPoint error = 0.0;
      const VoxelEvaluationResult result =
          computeVoxelError(gt_block.getVoxelByLinearIndex(linear_index),
                            test_block.getVoxelByLinearIndex(linear_index),
                            voxel_evaluation_mode, &error);

      switch (result) {
        case VoxelEvaluationResult::kEvaluated:
          total_squared_error += error * error;
          evaluation_details.min_error =
              std::min(evaluation_details.min_error, std::abs(error));
          evaluation_details.max_error =
              std::max(evaluation_details.max_error, std::abs(error));
          ++evaluation_details.num_evaluated_voxels;
          ++evaluation_details.num_overlapping_voxels;

          if (error_block) {
            VoxelType& error_voxel =
                error_block->getVoxelByLinearIndex(linear_index);
            setVoxelSdf<VoxelType>(std::abs(error), &error_voxel);
            setVoxelWeight<VoxelType>(1.0, &error_voxel);
          }

          break;
        case VoxelEvaluationResult::kIgnored:
          ++evaluation_details.num_ignored_voxels;
          ++evaluation_details.num_overlapping_voxels;
          break;
        case VoxelEvaluationResult::kNoOverlap:
          ++evaluation_details.num_non_overlapping_voxels;
          break;
        default:
          LOG(FATAL) << "Unkown voxel evaluation result: "
                     << static_cast<int>(result);
      }
    }
  }

  // Iterate over all blocks in the grond truth layer and look them up in the
  // test truth layer. This is only done to get the exact number of
  // non-overlapping voxels.
  BlockIndexList gt_block_list;
  layer_gt.getAllAllocatedBlocks(&gt_block_list);
  for (const BlockIndex& gt_block_index : gt_block_list) {
    const Block<VoxelType>& gt_block = layer_gt.getBlockByIndex(gt_block_index);
    if (!layer_test.hasBlock(gt_block_index)) {
      for (size_t linear_index = 0u; linear_index < num_voxels_per_block;
           ++linear_index) {
        const VoxelType& voxel = gt_block.getVoxelByLinearIndex(linear_index);
        if (isObservedVoxel(voxel)) {
          ++evaluation_details.num_non_overlapping_voxels;
        }
      }
    }
  }
```

(continues on next page)

(continued from previous page)

```
  // Return the RMSE.
  if (evaluation_details.num_evaluated_voxels == 0) {
    evaluation_details.rmse = 0.0;
  } else {
    evaluation_details.rmse =
        sqrt(total_squared_error / evaluation_details.num_evaluated_voxels);
  }

  // If the details are requested, output them.
  if (evaluation_result != nullptr) {
    *evaluation_result = evaluation_details;
  }

  VLOG(2) << evaluation_details.toString();

  return evaluation_details.rmse;
}

template <typename VoxelType>
FloatingPoint evaluateLayersRmse(const Layer<VoxelType>& layer_gt,
                                 const Layer<VoxelType>& layer_test) {
  return evaluateLayersRmse<VoxelType>(
      layer_gt, layer_test, VoxelEvaluationMode::kIgnoreErrorBehindTestSurface);
}

template <typename VoxelType>
VoxelEvaluationResult computeVoxelError(
    const VoxelType& voxel_gt, const VoxelType& voxel_test,
    const VoxelEvaluationMode evaluation_mode, FloatingPoint* error) {
  CHECK_NOTNULL(error);
  *error = 0.0;

  // Ignore voxels that are not observed in both layers.
  if (!isObservedVoxel(voxel_gt) || !isObservedVoxel(voxel_test)) {
    return VoxelEvaluationResult::kNoOverlap;
  }

  const bool ignore_behind_test_surface =
      (evaluation_mode == VoxelEvaluationMode::kIgnoreErrorBehindTestSurface) ||
      (evaluation_mode == VoxelEvaluationMode::kIgnoreErrorBehindAllSurfaces);

  const bool ignore_behind_gt_surface =
      (evaluation_mode == VoxelEvaluationMode::kIgnoreErrorBehindGtSurface) ||
      (evaluation_mode == VoxelEvaluationMode::kIgnoreErrorBehindAllSurfaces);

  if ((ignore_behind_test_surface && (voxel_test.distance) < 0.0) ||
      (ignore_behind_gt_surface && (voxel_gt.distance) < 0.0)) {
    return VoxelEvaluationResult::kIgnored;
  }

  *error = getVoxelSdf(voxel_test) - getVoxelSdf(voxel_gt);

  return VoxelEvaluationResult::kEvaluated;
}

template <>
```

(continues on next page)

(continued from previous page)

```
bool isObservedVoxel(const TsdfVoxel& voxel);
template <>
bool isObservedVoxel(const EsdfVoxel& voxel);
template <>
FloatingPoint getVoxelSdf(const TsdfVoxel& voxel);
template <>
FloatingPoint getVoxelSdf(const EsdfVoxel& voxel);
template <>
void setVoxelSdf(const FloatingPoint sdf, TsdfVoxel* voxel);
template <>
void setVoxelSdf(const FloatingPoint sdf, EsdfVoxel* voxel);
template <>
void setVoxelWeight(const FloatingPoint weight, TsdfVoxel* voxel);
template <>
void setVoxelWeight(const FloatingPoint weight, EsdfVoxel* voxel);

}  // namespace utils
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_EVALUATION_UTILS_H_
```

Includes

- algorithm
- string
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Included By

- *File interpolator_inl.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::utils*

Classes

- *Struct VoxelEvaluationDetails*

File icp.h

Contents

- *Definition (voxblox/include/voxblox/alignment/icp.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/alignment/icp.h)

Program Listing for File icp.h

Return to documentation for file (voxblox/include/voxblox/alignment/icp.h)

```
/*
 * Software License Agreement (BSD License)
 *
 *  Point Cloud Library (PCL) - www.pointclouds.org
 *  Copyright (c) 2010, Willow Garage, Inc.
 *  Copyright (c) 2012-, Open Perception, Inc.
 *
 *  All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions
 *  are met:
 *
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above
 *     copyright notice, this list of conditions and the following
 *     disclaimer in the documentation and/or other materials provided
 *     with the distribution.
 *   * Neither the name of the copyright holder(s) nor the names of its
 *     contributors may be used to endorse or promote products derived
 *     from this software without specific prior written permission.
 *
 *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 *  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 *  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 *  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 *  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 *  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 *  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 *  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 *  POSSIBILITY OF SUCH DAMAGE.
 *
 * $Id$
 *
 */
#ifndef VOXBLOX_ALIGNMENT_ICP_H_
```

(continues on next page)

(continued from previous page)

```
#define VOXBLOX_ALIGNMENT_ICP_H_

#include <algorithm>
#include <memory>
#include <thread>

#include "voxblox/core/block_hash.h"
#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/interpolator/interpolator.h"
#include "voxblox/utils/approx_hash_array.h"

namespace voxblox {

class ICP {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    bool refine_roll_pitch = false;
    int mini_batch_size = 20;
    FloatingPoint min_match_ratio = 0.8;
    FloatingPoint subsample_keep_ratio = 0.5;
    FloatingPoint inital_translation_weighting = 100.0;
    FloatingPoint inital_rotation_weighting = 100.0;
    size_t num_threads = std::thread::hardware_concurrency();
  };

  explicit ICP(const Config& config);

  size_t runICP(const Layer<TsdfVoxel>& tsdf_layer, const Pointcloud& points,
                const Transformation& inital_T_tsdf_sensor,
                Transformation* refined_T_tsdf_sensor,
                const unsigned seed = std::chrono::system_clock::now()
                                          .time_since_epoch()
                                          .count());

  bool refiningRollPitch() { return config_.refine_roll_pitch; }

 private:
  typedef Transformation::Vector6 Vector6;

  template <size_t dim>
  static bool getRotationFromMatchedPoints(const PointsMatrix& src_demean,
                                           const PointsMatrix& tgt_demean,
                                           Rotation* R_tgt_src) {
    static_assert((dim == 3) || (dim == 2),
                  "Rotation calculation is only meaningful for 2D or 3D data");
    CHECK_NOTNULL(R_tgt_src);

    SquareMatrix<3> rotation_matrix = SquareMatrix<3>::Identity();

    SquareMatrix<dim> H =
        src_demean.topRows<dim>() * tgt_demean.topRows<dim>().transpose();

    // Compute the Singular Value Decomposition
```

(continues on next page)

(continued from previous page)

```
    Eigen::JacobiSVD<SquareMatrix<dim>> svd(
        H, Eigen::ComputeFullU | Eigen::ComputeFullV);
    SquareMatrix<dim> u = svd.matrixU();
    SquareMatrix<dim> v = svd.matrixV();

    // Compute R = V * U'
    if (u.determinant() * v.determinant() < 0.0) {
      v.col(dim - 1) *= -1.0;
    }

    rotation_matrix.topLeftCorner<dim, dim>() = v * u.transpose();

    *R_tgt_src = Rotation(rotation_matrix);

    // not caught by is valid check
    if (!std::isfinite(rotation_matrix.sum())) {
      return false;
    }

    return Rotation::isValidRotationMatrix(rotation_matrix);
  }

  static bool getTransformFromMatchedPoints(const PointsMatrix& src,
                                            const PointsMatrix& tgt,
                                            const bool refine_roll_pitch,
                                            Transformation* T_tsdf_sensor);

  static void addNormalizedPointInfo(const Point& point,
                                     const Point& normalized_point_normal,
                                     Vector6* info_vector);

  void matchPoints(const Pointcloud& points, const size_t start_idx,
                   const Transformation& T_tsdf_sensor, PointsMatrix* src,
                   PointsMatrix* tgt, Vector6* info_vector);

  bool stepICP(const Pointcloud& points, const size_t start_idx,
               const Transformation& inital_T_tsdf_sensor,
               Transformation* refined_T_tsdf_sensor, Vector6* info_vector);

  void runThread(const Pointcloud& points,
                 Transformation* current_T_tsdf_sensor,
                 Vector6* base_info_vector, size_t* num_updates);

  Config config_;

  std::atomic<size_t> atomic_idx_;
  std::mutex mutex_;

  FloatingPoint voxel_size_;
  FloatingPoint voxel_size_inv_;
  std::shared_ptr<Interpolator<TsdfVoxel>> interpolator_;
};

}  // namespace voxblox

#endif  // VOXBLOX_ALIGNMENT_ICP_H_
```

Includes

- `algorithm`
- `memory`
- `thread`
- `voxblox/core/block_hash.h` (*File block_hash.h*)
- `voxblox/core/common.h` (*File common.h*)
- `voxblox/core/layer.h` (*File layer.h*)
- `voxblox/integrator/integrator_utils.h` (*File integrator_utils.h*)
- `voxblox/interpolator/interpolator.h` (*File interpolator.h*)
- `voxblox/utils/approx_hash_array.h` (*File approx_hash_array.h*)

Included By

- *File ros_params.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct ICP::Config*
- *Class ICP*

File integrator_utils.h

Contents

- *Definition (`voxblox/include/voxblox/integrator/integrator_utils.h`)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/integrator/integrator_utils.h)

Program Listing for File integrator_utils.h

Return to documentation for file (voxblox/include/voxblox/integrator/integrator_utils.h)

```
#ifndef VOXBLOX_INTEGRATOR_INTEGRATOR_UTILS_H_
#define VOXBLOX_INTEGRATOR_INTEGRATOR_UTILS_H_

#include <algorithm>
#include <array>
#include <atomic>
#include <vector>

#include <glog/logging.h>
#include <Eigen/Core>

#include "voxblox/core/block_hash.h"
#include "voxblox/core/common.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

class ThreadSafeIndex {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  explicit ThreadSafeIndex(size_t number_of_points);

  bool getNextIndex(size_t* idx);

  void reset();

 private:
  size_t getMixedIndex(size_t base_idx);

  std::atomic<size_t> atomic_idx_;
  const size_t number_of_points_;
  const size_t number_of_groups_;

  static constexpr size_t num_bits = 10;
  static constexpr size_t step_size_ = 1 << num_bits;
  static constexpr size_t bit_mask_ = step_size_ - 1;

  static const std::array<size_t, step_size_> offset_lookup_;
};

class RayCaster {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  RayCaster(const Point& origin, const Point& point_G,
            const bool is_clearing_ray, const bool voxel_carving_enabled,
            const FloatingPoint max_ray_length_m,
            const FloatingPoint voxel_size_inv,
            const FloatingPoint truncation_distance,
            const bool cast_from_origin = true);
```

(continues on next page)

(continued from previous page)

```
  RayCaster(const Point& start_scaled, const Point& end_scaled);

  bool nextRayIndex(GlobalIndex* ray_index);

 private:
  void setupRayCaster(const Point& start_scaled, const Point& end_scaled);

  Ray t_to_next_boundary_;
  GlobalIndex curr_index_;
  AnyIndex ray_step_signs_;
  Ray t_step_size_;

  uint ray_length_in_steps_;
  uint current_step_;
};

inline void castRay(const Point& start_scaled, const Point& end_scaled,
                    AlignedVector<GlobalIndex>* indices) {
  CHECK_NOTNULL(indices);

  RayCaster ray_caster(start_scaled, end_scaled);

  GlobalIndex ray_index;
  while (ray_caster.nextRayIndex(&ray_index)) {
    indices->push_back(ray_index);
  }
}

inline void getHierarchicalIndexAlongRay(
    const Point& start, const Point& end, size_t voxels_per_side,
    FloatingPoint voxel_size, FloatingPoint truncation_distance,
    bool voxel_carving_enabled, HierarchicalIndexMap* hierarchical_idx_map) {
  hierarchical_idx_map->clear();

  FloatingPoint voxels_per_side_inv = 1.0 / voxels_per_side;
  FloatingPoint voxel_size_inv = 1.0 / voxel_size;

  const Ray unit_ray = (end - start).normalized();

  const Point ray_end = end + unit_ray * truncation_distance;
  const Point ray_start =
      voxel_carving_enabled ? start : (end - unit_ray * truncation_distance);

  const Point start_scaled = ray_start * voxel_size_inv;
  const Point end_scaled = ray_end * voxel_size_inv;

  AlignedVector<GlobalIndex> global_voxel_index;
  timing::Timer cast_ray_timer("integrate/cast_ray");
  castRay(start_scaled, end_scaled, &global_voxel_index);
  cast_ray_timer.Stop();

  timing::Timer create_index_timer("integrate/create_hi_index");
  for (const GlobalIndex& global_voxel_idx : global_voxel_index) {
    BlockIndex block_idx = getBlockIndexFromGlobalVoxelIndex(
        global_voxel_idx, voxels_per_side_inv);
    VoxelIndex local_voxel_idx =
```

(continues on next page)

(continued from previous page)

```
        getLocalFromGlobalVoxelIndex(global_voxel_idx, voxels_per_side);

    if (local_voxel_idx.x() < 0) {
      local_voxel_idx.x() += voxels_per_side;
    }
    if (local_voxel_idx.y() < 0) {
      local_voxel_idx.y() += voxels_per_side;
    }
    if (local_voxel_idx.z() < 0) {
      local_voxel_idx.z() += voxels_per_side;
    }

    (*hierarchical_idx_map)[block_idx].push_back(local_voxel_idx);
  }
  create_index_timer.Stop();
}

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_INTEGRATOR_UTILS_H_
```

Includes

- `Eigen/Core`
- `algorithm`
- `array` (*File approx_hash_array.h*)
- `atomic`
- `glog/logging.h`
- `vector`
- `voxblox/core/block_hash.h` (*File block_hash.h*)
- `voxblox/core/common.h` (*File common.h*)
- `voxblox/utils/timing.h` (*File timing.h*)

Included By

- *File icp.h*
- *File esdf_integrator.h*
- *File esdf_occ_integrator.h*
- *File intensity_integrator.h*
- *File occupancy_integrator.h*
- *File tsdf_integrator.h*
- *File mesh_integrator.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class RayCaster*
- *Class ThreadSafeIndex*

File intensity_integrator.h

Contents

- *Definition (voxblox/include/voxblox/integrator/intensity_integrator.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/integrator/intensity_integrator.h)

Program Listing for File intensity_integrator.h

Return to documentation for file (voxblox/include/voxblox/integrator/intensity_integrator.h)

```
#ifndef VOXBLOX_INTEGRATOR_INTENSITY_INTEGRATOR_H_
#define VOXBLOX_INTEGRATOR_INTENSITY_INTEGRATOR_H_

#include <glog/logging.h>
#include <Eigen/Core>
#include <algorithm>
#include <queue>
#include <utility>
#include <vector>

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

class IntensityIntegrator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
```

(continues on next page)

(continued from previous page)

```
  IntensityIntegrator(const Layer<TsdfVoxel>& tsdf_layer,
                      Layer<IntensityVoxel>* intensity_layer);

  void setMaxDistance(const FloatingPoint max_distance) {
    max_distance_ = max_distance;
  }
  FloatingPoint getMaxDistance() const { return max_distance_; }

  void addIntensityBearingVectors(const Point& origin,
                                  const Pointcloud& bearing_vectors,
                                  const std::vector<float>& intensities);

 private:
  FloatingPoint max_distance_;
  float max_weight_;
  int intensity_prop_voxel_radius_;

  const Layer<TsdfVoxel>& tsdf_layer_;
  Layer<IntensityVoxel>* intensity_layer_;
};

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_INTENSITY_INTEGRATOR_H_
```

Includes

- Eigen/Core
- algorithm
- glog/logging.h
- queue (*File bucket_queue.h*)
- utility
- vector
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/integrator_utils.h (*File integrator_utils.h*)
- voxblox/utils/timing.h (*File timing.h*)

Included By

- *File intensity_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class IntensityIntegrator*

File intensity_server.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/intensity_server.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox_ros/include/voxblox_ros/intensity_server.h)

Program Listing for File intensity_server.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/intensity_server.h)

```
#ifndef VOXBLOX_ROS_INTENSITY_SERVER_H_
#define VOXBLOX_ROS_INTENSITY_SERVER_H_

#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/Image.h>
#include <memory>

#include <voxblox/core/voxel.h>
#include <voxblox/integrator/intensity_integrator.h>
#include <voxblox/utils/color_maps.h>

#include "voxblox_ros/intensity_vis.h"
#include "voxblox_ros/tsdf_server.h"

namespace voxblox {

class IntensityServer : public TsdfServer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  IntensityServer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private);
  virtual ~IntensityServer() {}

  virtual void updateMesh();
  virtual void publishPointclouds();

  void intensityImageCallback(const sensor_msgs::ImageConstPtr& image);

 protected:
  ros::Subscriber intensity_image_sub_;
```

(continues on next page)

(continued from previous page)

```
  // Publish markers for visualization.
  ros::Publisher intensity_pointcloud_pub_;
  ros::Publisher intensity_mesh_pub_;

  double focal_length_px_;

  int subsample_factor_;

  // Intensity layer, integrator, and color maps, all related to storing
  // and visualizing intensity data.
  std::shared_ptr<Layer<IntensityVoxel>> intensity_layer_;
  std::unique_ptr<IntensityIntegrator> intensity_integrator_;

  // Visualization tools.
  std::shared_ptr<ColorMap> color_map_;
};

}  // namespace voxblox

#endif  // VOXBLOX_ROS_INTENSITY_SERVER_H_
```

Includes

- cv_bridge/cv_bridge.h
- memory
- sensor_msgs/Image.h
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/intensity_integrator.h (*File intensity_integrator.h*)
- voxblox/utils/color_maps.h (*File color_maps.h*)
- voxblox_ros/intensity_vis.h (*File intensity_vis.h*)
- voxblox_ros/tsdf_server.h (*File tsdf_server.h*)

Namespaces

- *Namespace voxblox*

Classes

- *Class IntensityServer*

File intensity_vis.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/intensity_vis.h)*

- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/intensity_vis.h)

Program Listing for File intensity_vis.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/intensity_vis.h)

```
#ifndef VOXBLOX_ROS_INTENSITY_VIS_H_
#define VOXBLOX_ROS_INTENSITY_VIS_H_

#include <memory>

#include <voxblox/utils/color_maps.h>
#include <voxblox_msgs/Mesh.h>

#include "voxblox_ros/conversions.h"
#include "voxblox_ros/mesh_vis.h"

namespace voxblox {

inline void recolorVoxbloxMeshMsgByIntensity(
    const Layer<IntensityVoxel>& intensity_layer,
    const std::shared_ptr<ColorMap>& color_map, voxblox_msgs::Mesh* mesh_msg) {
  CHECK_NOTNULL(mesh_msg);
  CHECK(color_map);

  // Go over all the blocks in the mesh.
  for (voxblox_msgs::MeshBlock& mesh_block : mesh_msg->mesh_blocks) {
    // Look up verticies in the thermal layer.
    for (size_t vert_idx = 0u; vert_idx < mesh_block.x.size(); ++vert_idx) {
      // only needed if color information was originally missing
      mesh_block.r.resize(mesh_block.x.size());
      mesh_block.g.resize(mesh_block.x.size());
      mesh_block.b.resize(mesh_block.x.size());

      const IntensityVoxel* voxel = intensity_layer.getVoxelPtrByCoordinates(
          Point(mesh_block.x[vert_idx], mesh_block.y[vert_idx],
                mesh_block.z[vert_idx]));
      if (voxel != nullptr && voxel->weight > 0.0) {
        float intensity = voxel->intensity;
        Color new_color = color_map->colorLookup(intensity);
        mesh_block.r[vert_idx] = new_color.r;
        mesh_block.g[vert_idx] = new_color.g;
        mesh_block.b[vert_idx] = new_color.b;
      }
    }
  }
}

}  // namespace voxblox
```

(continues on next page)

(continued from previous page)

```
#endif  // VOXBLOX_ROS_INTENSITY_VIS_H_
```

Includes

- `memory`
- `voxblox/utils/color_maps.h` (*File color_maps.h*)
- `voxblox_msgs/Mesh.h`
- `voxblox_ros/conversions.h` (*File conversions.h*)
- `voxblox_ros/mesh_vis.h` (*File mesh_vis.h*)

Included By

- *File intensity_server.h*

Namespaces

- *Namespace voxblox*

File interactive_slider.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/interactive_slider.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox_ros/include/voxblox_ros/interactive_slider.h)

Program Listing for File interactive_slider.h

Return to documentation for file (`voxblox_ros/include/voxblox_ros/interactive_slider.h`)

```
#ifndef VOXBLOX_ROS_INTERACTIVE_SLIDER_H_
#define VOXBLOX_ROS_INTERACTIVE_SLIDER_H_

#include <functional>
#include <string>

#include <interactive_markers/interactive_marker_server.h>
#include <visualization_msgs/InteractiveMarkerFeedback.h>
```

(continues on next page)

(continued from previous page)

```
#include <voxblox/core/common.h>

namespace voxblox {

class InteractiveSlider {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  InteractiveSlider(
      const std::string& slider_name,
      const std::function<void(const double& slice_level)>& slider_callback,
      const Point& initial_position, const unsigned int free_plane_index,
      const float marker_scale_meters);
  virtual ~InteractiveSlider() {}

 private:
  const unsigned int free_plane_index_;
  interactive_markers::InteractiveMarkerServer interactive_marker_server_;

  virtual void interactiveMarkerFeedback(
      const visualization_msgs::InteractiveMarkerFeedbackConstPtr& feedback,
      const std::function<void(const double slice_level)>& slider_callback);
};

}  // namespace voxblox

#endif  // VOXBLOX_ROS_INTERACTIVE_SLIDER_H_
```

Includes

- functional
- interactive_markers/interactive_marker_server.h
- string
- visualization_msgs/InteractiveMarkerFeedback.h
- voxblox/core/common.h (*File common.h*)

Namespaces

- *Namespace voxblox*

Classes

- *Class InteractiveSlider*

File interpolator.h

Contents

- *Definition (voxblox/include/voxblox/interpolator/interpolator.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/interpolator/interpolator.h)

Program Listing for File interpolator.h

Return to documentation for file (voxblox/include/voxblox/interpolator/interpolator.h)

```
#ifndef VOXBLOX_INTERPOLATOR_INTERPOLATOR_H_
#define VOXBLOX_INTERPOLATOR_INTERPOLATOR_H_

#include <memory>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {

template <typename VoxelType>
class Interpolator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<Interpolator> Ptr;

  explicit Interpolator(const Layer<VoxelType>* layer);

  bool getGradient(const Point& pos, Point* grad,
                   const bool interpolate = false) const;

  bool getDistance(const Point& pos, FloatingPoint* distance,
                   bool interpolate = false) const;

  bool getVoxel(const Point& pos, VoxelType* voxel,
                bool interpolate = false) const;

  bool getAdaptiveDistanceAndGradient(const Point& pos, FloatingPoint* distance,
                                      Point* grad) const;

  bool getNearestDistanceAndWeight(const Point& pos, FloatingPoint* distance,
                                   float* weight) const;

 private:
  bool setIndexes(const Point& pos, BlockIndex* block_index,
                  InterpIndexes* voxel_indexes) const;
```

(continues on next page)

(continued from previous page)

```
  void getQVector(const Point& voxel_pos, const Point& pos,
                  const FloatingPoint voxel_size_inv,
                  InterpVector* q_vector) const;

  bool getVoxelsAndQVector(const BlockIndex& block_index,
                           const InterpIndexes& voxel_indexes, const Point& pos,
                           const VoxelType** voxels,
                           InterpVector* q_vector) const;

  bool getVoxelsAndQVector(const Point& pos, const VoxelType** voxels,
                           InterpVector* q_vector) const;

  bool getInterpDistance(const Point& pos, FloatingPoint* distance) const;

  bool getNearestDistance(const Point& pos, FloatingPoint* distance) const;

  bool getInterpVoxel(const Point& pos, VoxelType* voxel) const;

  bool getNearestVoxel(const Point& pos, VoxelType* voxel) const;

  static float getVoxelSdf(const VoxelType& voxel);
  static float getVoxelWeight(const VoxelType& voxel);

  static uint8_t getRed(const VoxelType& voxel);
  static uint8_t getBlue(const VoxelType& voxel);
  static uint8_t getGreen(const VoxelType& voxel);
  static uint8_t getAlpha(const VoxelType& voxel);

  template <typename TGetter>
  static FloatingPoint interpMember(const InterpVector& q_vector,
                                    const VoxelType** voxels,
                                    TGetter (*getter)(const VoxelType&));

  static VoxelType interpVoxel(const InterpVector& q_vector,
                               const VoxelType** voxels);

  const Layer<VoxelType>* layer_;
};

}  // namespace voxblox

#endif  // VOXBLOX_INTERPOLATOR_INTERPOLATOR_H_

#include "voxblox/interpolator/interpolator_inl.h"
```

Includes

- memory
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/interpolator/interpolator_inl.h (*File interpolator_inl.h*)

Included By

- *File icp.h*
- *File esdf_map.h*
- *File merge_integration.h*
- *File mesh_integrator.h*

Namespaces

- *Namespace voxblox*

Classes

- *Template Class Interpolator*

File interpolator_inl.h

Contents

- *Definition (voxblox/include/voxblox/interpolator/interpolator_inl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/interpolator/interpolator_inl.h)

Program Listing for File interpolator_inl.h

Return to documentation for file (voxblox/include/voxblox/interpolator/interpolator_inl.h)

```
#ifndef VOXBLOX_INTERPOLATOR_INTERPOLATOR_INL_H_
#define VOXBLOX_INTERPOLATOR_INTERPOLATOR_INL_H_

#include <iostream>

#include "voxblox/utils/evaluation_utils.h"

namespace voxblox {

template <typename VoxelType>
Interpolator<VoxelType>::Interpolator(const Layer<VoxelType>* layer)
    : layer_(layer) {}

template <typename VoxelType>
bool Interpolator<VoxelType>::getDistance(const Point& pos,
```

(continues on next page)

(continued from previous page)

```
                                          FloatingPoint* distance,
                                          bool interpolate) const {
  if (interpolate) {
    return getInterpDistance(pos, distance);
  } else {
    return getNearestDistance(pos, distance);
  }
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getVoxel(const Point& pos, VoxelType* voxel,
                                       bool interpolate) const {
  if (interpolate) {
    return getInterpVoxel(pos, voxel);
  } else {
    return getNearestVoxel(pos, voxel);
  }
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getGradient(const Point& pos, Point* grad,
                                          const bool interpolate) const {
  CHECK_NOTNULL(grad);

  typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
      layer_->getBlockPtrByCoordinates(pos);
  if (block_ptr == nullptr) {
    return false;
  }
  // Now get the gradient.
  *grad = Point::Zero();
  // Iterate over all 3 D, and over negative and positive signs in central
  // difference.
  for (unsigned int i = 0u; i < 3u; ++i) {
    for (int sign = -1; sign <= 1; sign += 2) {
      Point offset = Point::Zero();
      offset(i) = sign * block_ptr->voxel_size();
      FloatingPoint offset_distance;
      if (!getDistance(pos + offset, &offset_distance, interpolate)) {
        return false;
      }
      (*grad)(i) += offset_distance * static_cast<FloatingPoint>(sign);
    }
  }
  // Scale by correct size.
  // This is central difference, so it's 2x voxel size between measurements.
  *grad /= (2 * block_ptr->voxel_size());
  return true;
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getAdaptiveDistanceAndGradient(
    const Point& pos, FloatingPoint* distance, Point* grad) const {
  // TODO(helenol): try to see how to minimize number of lookups for the
  // gradient and interpolation calculations...

  // Get the nearest neighbor distance first, we need this for the gradient
```

(continues on next page)

(continued from previous page)

```
  // calculations anyway.
  FloatingPoint nearest_neighbor_distance = 0.0f;
  bool interpolate = false;
  if (!getDistance(pos, &nearest_neighbor_distance, interpolate)) {
    // Then there is no data here at all.
    return false;
  }

  // Then try to get the interpolated distance.
  interpolate = true;
  bool has_interpolated_distance = getDistance(pos, distance, interpolate);

  // Now try to estimate the gradient. Same general procedure as getGradient()
  // above, but also allow finite difference methods other than central
  // difference (left difference, right difference).
  typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
      layer_->getBlockPtrByCoordinates(pos);
  if (block_ptr == nullptr) {
    return false;
  }

  Point gradient = Point::Zero();

  // Try to get the full gradient if possible.
  bool has_interpolated_gradient = false;
  if (has_interpolated_distance) {
    has_interpolated_gradient = getGradient(pos, &gradient, interpolate);
  }

  if (!has_interpolated_gradient) {
    // Otherwise fall back to this...
    interpolate = false;
    for (unsigned int i = 0u; i < 3u; ++i) {
      // First check if we can get both sides for central difference.
      Point offset = Point::Zero();
      offset(i) = block_ptr->voxel_size();
      FloatingPoint left_distance = 0.0f, right_distance = 0.0f;
      bool left_valid = getDistance(pos - offset, &left_distance, interpolate);
      bool right_valid =
          getDistance(pos + offset, &right_distance, interpolate);

      if (left_valid && right_valid) {
        gradient(i) =
            (right_distance - left_distance) / (2.0f * block_ptr->voxel_size());
      } else if (left_valid) {
        gradient(i) = (nearest_neighbor_distance - left_distance) /
                      block_ptr->voxel_size();
      } else if (right_valid) {
        gradient(i) = (right_distance - nearest_neighbor_distance) /
                      block_ptr->voxel_size();
      } else {
        // This has no neighbors on any side in this dimension :(
        return false;
      }
    }
  }
```

(continues on next page)

(continued from previous page)

```
  // If we weren't able to get the original interpolated distance value, then
  // use the computed gradient to estimate what the value should be.
  if (!has_interpolated_distance) {
    VoxelIndex voxel_index =
        block_ptr->computeTruncatedVoxelIndexFromCoordinates(pos);
    Point voxel_pos = block_ptr->computeCoordinatesFromVoxelIndex(voxel_index);

    Point voxel_offset = pos - voxel_pos;
    *distance = nearest_neighbor_distance + voxel_offset.dot(gradient);
  }

  *grad = gradient;
  return true;
}

template <typename VoxelType>
bool Interpolator<VoxelType>::setIndexes(const Point& pos,
                                         BlockIndex* block_index,
                                         InterpIndexes* voxel_indexes) const {
  // get voxel index
  *block_index = layer_->computeBlockIndexFromCoordinates(pos);
  typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
      layer_->getBlockPtrByIndex(*block_index);
  if (block_ptr == nullptr) {
    return false;
  }
  VoxelIndex voxel_index =
      block_ptr->computeTruncatedVoxelIndexFromCoordinates(pos);

  // shift index to bottom left corner voxel (makes math easier)
  Point center_offset =
      pos - block_ptr->computeCoordinatesFromVoxelIndex(voxel_index);
  for (size_t i = 0; i < static_cast<size_t>(center_offset.rows()); ++i) {
    // ensure that the point we are interpolating to is always larger than the
    // center of the voxel_index in all dimensions
    if (center_offset(i) < 0) {
      voxel_index(i)--;
      // move blocks if needed
      if (voxel_index(i) < 0) {
        (*block_index)(i)--;
        voxel_index(i) += block_ptr->voxels_per_side();
      }
    }
  }

  // get indexes of neighbors

  // FROM PAPER (http://spie.org/samples/PM159.pdf)
  // clang-format off
  (*voxel_indexes) <<
    0, 0, 0, 0, 1, 1, 1, 1,
    0, 0, 1, 1, 0, 0, 1, 1,
    0, 1, 0, 1, 0, 1, 0, 1;
  // clang-format on

  voxel_indexes->colwise() += voxel_index.array();
  return true;
```

(continues on next page)

(continued from previous page)

```
}

template <typename VoxelType>
void Interpolator<VoxelType>::getQVector(const Point& voxel_pos,
                                         const Point& pos,
                                         const FloatingPoint voxel_size_inv,
                                         InterpVector* q_vector) const {
  CHECK_NOTNULL(q_vector);

  const Point voxel_offset = (pos - voxel_pos) * voxel_size_inv;

  CHECK((voxel_offset.array() >= 0).all());  // NOLINT

  // FROM PAPER (http://spie.org/samples/PM159.pdf)
  // clang-format off
  *q_vector <<
      1,
      voxel_offset[0],
      voxel_offset[1],
      voxel_offset[2],
      voxel_offset[0] * voxel_offset[1],
      voxel_offset[1] * voxel_offset[2],
      voxel_offset[2] * voxel_offset[0],
      voxel_offset[0] * voxel_offset[1] * voxel_offset[2];
  // clang-format on
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getVoxelsAndQVector(
    const BlockIndex& block_index, const InterpIndexes& voxel_indexes,
    const Point& pos, const VoxelType** voxels, InterpVector* q_vector) const {
  CHECK_NOTNULL(q_vector);

  // for each voxel index
  for (size_t i = 0; i < static_cast<size_t>(voxel_indexes.cols()); ++i) {
    typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
        layer_->getBlockPtrByIndex(block_index);
    if (block_ptr == nullptr) {
      return false;
    }

    VoxelIndex voxel_index = voxel_indexes.col(i);
    // if voxel index is too large get neighboring block and update index
    if ((voxel_index.array() >= block_ptr->voxels_per_side()).any()) {
      BlockIndex new_block_index = block_index;
      for (size_t j = 0; j < static_cast<size_t>(block_index.rows()); ++j) {
        if (voxel_index(j) >=
            static_cast<IndexElement>(block_ptr->voxels_per_side())) {
          new_block_index(j)++;
          voxel_index(j) -= block_ptr->voxels_per_side();
        }
      }
      block_ptr = layer_->getBlockPtrByIndex(new_block_index);
      if (block_ptr == nullptr) {
        return false;
      }
    }
```

(continues on next page)

(continued from previous page)

```
    // use bottom left corner voxel to compute weights vector
    if (i == 0) {
      getQVector(block_ptr->computeCoordinatesFromVoxelIndex(voxel_index), pos,
                 block_ptr->voxel_size_inv(), q_vector);
    }

    const VoxelType& voxel = block_ptr->getVoxelByVoxelIndex(voxel_index);

    voxels[i] = &voxel;
    if (!utils::isObservedVoxel(voxel)) {
      return false;
    }
  }
  return true;
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getVoxelsAndQVector(
    const Point& pos, const VoxelType** voxels, InterpVector* q_vector) const {
  // get block and voxels indexes (some voxels may have negative indexes)
  BlockIndex block_index;
  InterpIndexes voxel_indexes;
  if (!setIndexes(pos, &block_index, &voxel_indexes)) {
    return false;
  }

  // get distances of 8 surrounding voxels and weights vector
  return getVoxelsAndQVector(block_index, voxel_indexes, pos, voxels, q_vector);
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getInterpDistance(const Point& pos,
                                                FloatingPoint* distance) const {
  CHECK_NOTNULL(distance);

  // get distances of 8 surrounding voxels and weights vector
  const VoxelType* voxels[8];
  InterpVector q_vector;
  if (!getVoxelsAndQVector(pos, voxels, &q_vector)) {
    return false;
  } else {
    *distance = interpMember(q_vector, voxels, &getVoxelSdf);
    return true;
  }
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getNearestDistance(
    const Point& pos, FloatingPoint* distance) const {
  CHECK_NOTNULL(distance);

  typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
      layer_->getBlockPtrByCoordinates(pos);
  if (block_ptr == nullptr) {
    return false;
  }
```

(continues on next page)

(continued from previous page)

```
  const VoxelType& voxel = block_ptr->getVoxelByCoordinates(pos);

  *distance = getVoxelSdf(voxel);

  return utils::isObservedVoxel(voxel);
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getInterpVoxel(const Point& pos,
                                             VoxelType* voxel) const {
  CHECK_NOTNULL(voxel);

  // get voxels of 8 surrounding voxels and weights vector
  const VoxelType* voxels[8];
  InterpVector q_vector;
  if (!getVoxelsAndQVector(pos, voxels, &q_vector)) {
    return false;
  } else {
    *voxel = interpVoxel(q_vector, voxels);
    return true;
  }
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getNearestVoxel(const Point& pos,
                                              VoxelType* voxel) const {
  CHECK_NOTNULL(voxel);

  typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
      layer_->getBlockPtrByCoordinates(pos);
  if (block_ptr == nullptr) {
    return false;
  }

  *voxel = block_ptr->getVoxelByCoordinates(pos);

  return utils::isObservedVoxel(*voxel);
}

template <typename VoxelType>
bool Interpolator<VoxelType>::getNearestDistanceAndWeight(
    const Point& pos, FloatingPoint* distance, float* weight) const {
  CHECK_NOTNULL(distance);
  CHECK_NOTNULL(weight);

  typename Layer<VoxelType>::BlockType::ConstPtr block_ptr =
      layer_->getBlockPtrByCoordinates(pos);
  if (block_ptr == nullptr) {
    return false;
  }
  const VoxelType& voxel = block_ptr->getVoxelByCoordinates(pos);
  *distance = getVoxelSdf(voxel);
  *weight = getVoxelWeight(voxel);
  return true;
}

template <>
```

(continues on next page)

(continued from previous page)

```
inline float Interpolator<TsdfVoxel>::getVoxelSdf(const TsdfVoxel& voxel) {
  return voxel.distance;
}

template <>
inline float Interpolator<EsdfVoxel>::getVoxelSdf(const EsdfVoxel& voxel) {
  return voxel.distance;
}

template <>
inline float Interpolator<TsdfVoxel>::getVoxelWeight(const TsdfVoxel& voxel) {
  return voxel.weight;
}

template <>
inline float Interpolator<EsdfVoxel>::getVoxelWeight(const EsdfVoxel& voxel) {
  return voxel.observed ? 1.0f : 0.0f;
}

template <>
inline uint8_t Interpolator<TsdfVoxel>::getRed(const TsdfVoxel& voxel) {
  return voxel.color.r;
}

template <>
inline uint8_t Interpolator<TsdfVoxel>::getGreen(const TsdfVoxel& voxel) {
  return voxel.color.g;
}

template <>
inline uint8_t Interpolator<TsdfVoxel>::getBlue(const TsdfVoxel& voxel) {
  return voxel.color.b;
}

template <>
inline uint8_t Interpolator<TsdfVoxel>::getAlpha(const TsdfVoxel& voxel) {
  return voxel.color.a;
}

template <typename VoxelType>
template <typename TGetter>
inline FloatingPoint Interpolator<VoxelType>::interpMember(
    const InterpVector& q_vector, const VoxelType** voxels,
    TGetter (*getter)(const VoxelType&)) {
  InterpVector data;
  for (int i = 0; i < data.size(); ++i) {
    data[i] = static_cast<FloatingPoint>((*getter)(*voxels[i]));
  }

  // FROM PAPER (http://spie.org/samples/PM159.pdf)
  // clang-format off
  static const InterpTable interp_table =
      (InterpTable() <<
        1,  0,  0,  0,  0,  0,  0,  0,
       -1,  0,  0,  0,  1,  0,  0,  0,
       -1,  0,  1,  0,  0,  0,  0,  0,
       -1,  1,  0,  0,  0,  0,  0,  0,
```

(continues on next page)

(continued from previous page)

```
        1,  0, -1,  0, -1,  0,  1,  0,
        1, -1, -1,  1,  0,  0,  0,  0,
        1, -1,  0,  0, -1,  1,  0,  0,
       -1,  1,  1, -1,  1, -1, -1,  1
       )
          .finished();
  // clang-format on
  return q_vector * (interp_table * data.transpose());
}

template <>
inline TsdfVoxel Interpolator<TsdfVoxel>::interpVoxel(
    const InterpVector& q_vector, const TsdfVoxel** voxels) {
  TsdfVoxel voxel;
  voxel.distance = interpMember(q_vector, voxels, &getVoxelSdf);
  voxel.weight = interpMember(q_vector, voxels, &getVoxelWeight);

  voxel.color.r = interpMember(q_vector, voxels, &getRed);
  voxel.color.g = interpMember(q_vector, voxels, &getGreen);
  voxel.color.b = interpMember(q_vector, voxels, &getBlue);
  voxel.color.a = interpMember(q_vector, voxels, &getAlpha);

  return voxel;
}

}  // namespace voxblox

#endif  // VOXBLOX_INTERPOLATOR_INTERPOLATOR_INL_H_
```

Includes

- `iostream`
- `voxblox/utils/evaluation_utils.h` (*File evaluation_utils.h*)

Included By

- *File interpolator.h*

Namespaces

- *Namespace voxblox*

File layer.h

Contents

- *Definition (`voxblox/include/voxblox/core/layer.h`)*
- *Includes*

- *Included By*
- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/core/layer.h`)

Program Listing for File layer.h

Return to documentation for file (`voxblox/include/voxblox/core/layer.h`)

```
#ifndef VOXBLOX_CORE_LAYER_H_
#define VOXBLOX_CORE_LAYER_H_

#include <glog/logging.h>
#include <memory>
#include <string>
#include <utility>

#include "./Block.pb.h"
#include "./Layer.pb.h"
#include "voxblox/core/block.h"
#include "voxblox/core/block_hash.h"
#include "voxblox/core/common.h"
#include "voxblox/core/voxel.h"

namespace voxblox {

template <typename VoxelType>
class Layer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<Layer> Ptr;
  typedef Block<VoxelType> BlockType;
  typedef
      typename AnyIndexHashMapType<typename BlockType::Ptr>::type BlockHashMap;
  typedef typename std::pair<BlockIndex, typename BlockType::Ptr> BlockMapPair;

  explicit Layer(FloatingPoint voxel_size, size_t voxels_per_side)
      : voxel_size_(voxel_size), voxels_per_side_(voxels_per_side) {
    CHECK_GT(voxel_size_, 0.0f);
    voxel_size_inv_ = 1.0 / voxel_size_;

    block_size_ = voxel_size_ * voxels_per_side_;
    CHECK_GT(block_size_, 0.0f);
    block_size_inv_ = 1.0 / block_size_;
    CHECK_GT(voxels_per_side_, 0u);
    voxels_per_side_inv_ = 1.0f / static_cast<FloatingPoint>(voxels_per_side_);
  }

  explicit Layer(const LayerProto& proto);

  explicit Layer(const Layer& other);
```

(continues on next page)

(continued from previous page)

```
  virtual ~Layer() {}

  enum class BlockMergingStrategy { kProhibit, kReplace, kDiscard, kMerge };

  inline const BlockType& getBlockByIndex(const BlockIndex& index) const {
    typename BlockHashMap::const_iterator it = block_map_.find(index);
    if (it == block_map_.end()) {
      LOG(FATAL) << "Accessed unallocated block at " << index.transpose();
    }
    return *(it->second);
  }

  inline BlockType& getBlockByIndex(const BlockIndex& index) {
    typename BlockHashMap::iterator it = block_map_.find(index);
    if (it == block_map_.end()) {
      LOG(FATAL) << "Accessed unallocated block at " << index.transpose();
    }
    return *(it->second);
  }

  inline typename BlockType::ConstPtr getBlockPtrByIndex(
      const BlockIndex& index) const {
    typename BlockHashMap::const_iterator it = block_map_.find(index);
    if (it != block_map_.end()) {
      return it->second;
    } else {
      return typename BlockType::ConstPtr();
    }
  }

  inline typename BlockType::Ptr getBlockPtrByIndex(const BlockIndex& index) {
    typename BlockHashMap::iterator it = block_map_.find(index);
    if (it != block_map_.end()) {
      return it->second;
    } else {
      return typename BlockType::Ptr();
    }
  }

  inline typename BlockType::Ptr allocateBlockPtrByIndex(
      const BlockIndex& index) {
    typename BlockHashMap::iterator it = block_map_.find(index);
    if (it != block_map_.end()) {
      return it->second;
    } else {
      return allocateNewBlock(index);
    }
  }

  inline typename BlockType::ConstPtr getBlockPtrByCoordinates(
      const Point& coords) const {
    return getBlockPtrByIndex(computeBlockIndexFromCoordinates(coords));
  }

  inline typename BlockType::Ptr getBlockPtrByCoordinates(const Point& coords) {
    return getBlockPtrByIndex(computeBlockIndexFromCoordinates(coords));
  }
```

(continues on next page)

(continued from previous page)

```
inline typename BlockType::Ptr allocateBlockPtrByCoordinates(
    const Point& coords) {
  return allocateBlockPtrByIndex(computeBlockIndexFromCoordinates(coords));
}

inline BlockIndex computeBlockIndexFromCoordinates(
    const Point& coords) const {
  return getGridIndexFromPoint<BlockIndex>(coords, block_size_inv_);
}

typename BlockType::Ptr allocateNewBlock(const BlockIndex& index) {
  auto insert_status = block_map_.emplace(
      index, std::make_shared<BlockType>(
                 voxels_per_side_, voxel_size_,
                 getOriginPointFromGridIndex(index, block_size_)));

  DCHECK(insert_status.second)
      << "Block already exists when allocating at " << index.transpose();

  DCHECK(insert_status.first->second);
  DCHECK_EQ(insert_status.first->first, index);
  return insert_status.first->second;
}

inline typename BlockType::Ptr allocateNewBlockByCoordinates(
    const Point& coords) {
  return allocateNewBlock(computeBlockIndexFromCoordinates(coords));
}

inline void insertBlock(
    const std::pair<const BlockIndex, typename Block<VoxelType>::Ptr>&
        block_pair) {
  auto insert_status = block_map_.insert(block_pair);

  DCHECK(insert_status.second) << "Block already exists when inserting at "
                               << insert_status.first->first.transpose();

  DCHECK(insert_status.first->second);
}

void removeBlock(const BlockIndex& index) { block_map_.erase(index); }
void removeAllBlocks() { block_map_.clear(); }

void removeBlockByCoordinates(const Point& coords) {
  block_map_.erase(computeBlockIndexFromCoordinates(coords));
}

void removeDistantBlocks(const Point& center, const double max_distance) {
  AlignedVector<BlockIndex> needs_erasing;
  for (const std::pair<const BlockIndex, typename BlockType::Ptr>& kv :
       block_map_) {
    if ((kv.second->origin() - center).squaredNorm() >
        max_distance * max_distance) {
      needs_erasing.push_back(kv.first);
    }
  }
```

(continues on next page)

(continued from previous page)

```
  for (const BlockIndex& index : needs_erasing) {
    block_map_.erase(index);
  }
}

void getAllAllocatedBlocks(BlockIndexList* blocks) const {
  CHECK_NOTNULL(blocks);
  blocks->clear();
  blocks->reserve(block_map_.size());
  for (const std::pair<const BlockIndex, typename BlockType::Ptr>& kv :
       block_map_) {
    blocks->emplace_back(kv.first);
  }
}

void getAllUpdatedBlocks(BlockIndexList* blocks) const {
  CHECK_NOTNULL(blocks);
  blocks->clear();
  for (const std::pair<const BlockIndex, typename BlockType::Ptr>& kv :
       block_map_) {
    if (kv.second->updated()) {
      blocks->emplace_back(kv.first);
    }
  }
}

size_t getNumberOfAllocatedBlocks() const { return block_map_.size(); }

bool hasBlock(const BlockIndex& block_index) const {
  return block_map_.count(block_index) > 0;
}

inline const VoxelType* getVoxelPtrByGlobalIndex(
    const GlobalIndex& global_voxel_index) const {
  const BlockIndex block_index = getBlockIndexFromGlobalVoxelIndex(
      global_voxel_index, voxels_per_side_inv_);
  if (!hasBlock(block_index)) {
    return nullptr;
  }
  const VoxelIndex local_voxel_index =
      getLocalFromGlobalVoxelIndex(global_voxel_index, voxels_per_side_);
  const Block<VoxelType>& block = getBlockByIndex(block_index);
  return &block.getVoxelByVoxelIndex(local_voxel_index);
}

inline VoxelType* getVoxelPtrByGlobalIndex(
    const GlobalIndex& global_voxel_index) {
  const BlockIndex block_index = getBlockIndexFromGlobalVoxelIndex(
      global_voxel_index, voxels_per_side_inv_);
  if (!hasBlock(block_index)) {
    return nullptr;
  }
  const VoxelIndex local_voxel_index =
      getLocalFromGlobalVoxelIndex(global_voxel_index, voxels_per_side_);
  Block<VoxelType>& block = getBlockByIndex(block_index);
  return &block.getVoxelByVoxelIndex(local_voxel_index);
}
```

(continues on next page)

(continued from previous page)

```
  inline const VoxelType* getVoxelPtrByCoordinates(const Point& coords) const {
    typename Block<VoxelType>::ConstPtr block_ptr =
        getBlockPtrByIndex(computeBlockIndexFromCoordinates(coords));
    if (!block_ptr) {
      return nullptr;
    }
    return block_ptr->getVoxelPtrByCoordinates(coords);
  }

  inline VoxelType* getVoxelPtrByCoordinates(const Point& coords) {
    typename Block<VoxelType>::Ptr block_ptr =
        getBlockPtrByIndex(computeBlockIndexFromCoordinates(coords));
    if (!block_ptr) {
      return nullptr;
    }
    return block_ptr->getVoxelPtrByCoordinates(coords);
  }

  FloatingPoint block_size() const { return block_size_; }
  FloatingPoint block_size_inv() const { return block_size_inv_; }
  FloatingPoint voxel_size() const { return voxel_size_; }
  FloatingPoint voxel_size_inv() const { return voxel_size_inv_; }
  size_t voxels_per_side() const { return voxels_per_side_; }
  FloatingPoint voxels_per_side_inv() const { return voxels_per_side_inv_; }

  // Serialization tools.
  void getProto(LayerProto* proto) const;
  bool isCompatible(const LayerProto& layer_proto) const;
  bool isCompatible(const BlockProto& layer_proto) const;
  bool saveToFile(const std::string& file_path, bool clear_file = true) const;
  // Default behavior is to clear the file.
  bool saveSubsetToFile(const std::string& file_path,
                        BlockIndexList blocks_to_include,
                        bool include_all_blocks, bool clear_file = true) const;
  bool addBlockFromProto(const BlockProto& block_proto,
                         BlockMergingStrategy strategy);

  size_t getMemorySize() const;

 protected:
  FloatingPoint voxel_size_;
  size_t voxels_per_side_;
  FloatingPoint block_size_;

  // Derived types.
  FloatingPoint voxel_size_inv_;
  FloatingPoint block_size_inv_;
  FloatingPoint voxels_per_side_inv_;

  std::string getType() const;

  BlockHashMap block_map_;
};

}  // namespace voxblox
```

(continues on next page)

(continued from previous page)

```
#include "voxblox/core/layer_inl.h"

#endif  // VOXBLOX_CORE_LAYER_H_
```

Includes

- ./Block.pb.h
- ./Layer.pb.h
- glog/logging.h
- memory
- string
- utility
- voxblox/core/block.h (*File block.h*)
- voxblox/core/block_hash.h (*File block_hash.h*)
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer_inl.h (*File layer_inl.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Included By

- *File icp.h*
- *File esdf_map.h*
- *File occupancy_map.h*
- *File tsdf_map.h*
- *File esdf_integrator.h*
- *File esdf_occ_integrator.h*
- *File intensity_integrator.h*
- *File merge_integration.h*
- *File occupancy_integrator.h*
- *File tsdf_integrator.h*
- *File interpolator.h*
- *File layer_io.h*
- *File sdf_ply.h*
- *File mesh_integrator.h*
- *File objects.h*
- *File simulation_world.h*
- *File layer_test_utils.h*

- *File distance_utils.h*
- *File evaluation_utils.h*
- *File layer_utils.h*
- *File neighbor_tools.h*
- *File planning_utils.h*
- *File planning_utils_inl.h*
- *File conversions.h*
- *File ptcloud_vis.h*

Namespaces

- *Namespace voxblox*

Classes

- *Template Class Layer*

File layer_inl.h

Contents

- *Definition (voxblox/include/voxblox/core/layer_inl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/core/layer_inl.h)

Program Listing for File layer_inl.h

Return to documentation for file (voxblox/include/voxblox/core/layer_inl.h)

```
#ifndef VOXBLOX_CORE_LAYER_INL_H_
#define VOXBLOX_CORE_LAYER_INL_H_

#include <fstream>  // NOLINT
#include <limits>
#include <string>
#include <utility>

#include <glog/logging.h>
#include <google/protobuf/io/coded_stream.h>
#include <google/protobuf/io/zero_copy_stream.h>
```

(continues on next page)

(continued from previous page)

```
#include <google/protobuf/io/zero_copy_stream_impl.h>
#include <google/protobuf/message.h>
#include <google/protobuf/message_lite.h>

#include "./Block.pb.h"
#include "./Layer.pb.h"
#include "voxblox/core/block.h"
#include "voxblox/core/voxel.h"
#include "voxblox/utils/protobuf_utils.h"

namespace voxblox {

template <typename VoxelType>
Layer<VoxelType>::Layer(const LayerProto& proto)
    : voxel_size_(proto.voxel_size()),
      voxels_per_side_(proto.voxels_per_side()) {
  CHECK_EQ(getType().compare(proto.type()), 0)
      << "Incorrect voxel type, proto type: " << proto.type()
      << " layer type: " << getType();

  // Derived config parameter.
  CHECK_GT(proto.voxel_size(), 0.0);
  block_size_ = voxel_size_ * voxels_per_side_;
  CHECK_GT(block_size_, 0.0);
  block_size_inv_ = 1.0 / block_size_;
  CHECK_GT(proto.voxels_per_side(), 0u);
  voxels_per_side_inv_ = 1.0f / static_cast<FloatingPoint>(voxels_per_side_);
}

template <typename VoxelType>
void Layer<VoxelType>::getProto(LayerProto* proto) const {
  CHECK_NOTNULL(proto);

  CHECK_NE(getType().compare(voxel_types::kNotSerializable), 0)
      << "The voxel type of this layer is not serializable!";

  proto->set_voxel_size(voxel_size_);
  proto->set_voxels_per_side(voxels_per_side_);
  proto->set_type(getType());
}

template <typename VoxelType>
Layer<VoxelType>::Layer(const Layer& other) {
  voxel_size_ = other.voxel_size_;
  voxels_per_side_ = other.voxels_per_side_;
  block_size_ = other.block_size_;
  block_size_inv_ = other.block_size_inv_;

  for (const typename BlockHashMap::value_type& key_value_pair :
       other.block_map_) {
    const BlockIndex& block_idx = key_value_pair.first;
    const typename BlockType::Ptr& block_ptr = key_value_pair.second;
    CHECK(block_ptr);

    typename BlockType::Ptr new_block = allocateBlockPtrByIndex(block_idx);
    CHECK(new_block);
```

(continues on next page)

(continued from previous page)

```
    for (size_t linear_idx = 0u; linear_idx < block_ptr->num_voxels();
         ++linear_idx) {
      const VoxelType& voxel = block_ptr->getVoxelByLinearIndex(linear_idx);
      VoxelType& new_voxel = new_block->getVoxelByLinearIndex(linear_idx);
      new_voxel = voxel;
    }
  }
}

template <typename VoxelType>
bool Layer<VoxelType>::saveToFile(const std::string& file_path,
                                  bool clear_file) const {
  constexpr bool kIncludeAllBlocks = true;
  return saveSubsetToFile(file_path, BlockIndexList(), kIncludeAllBlocks,
                          clear_file);
}

template <typename VoxelType>
bool Layer<VoxelType>::saveSubsetToFile(const std::string& file_path,
                                        BlockIndexList blocks_to_include,
                                        bool include_all_blocks,
                                        bool clear_file) const {
  CHECK_NE(getType().compare(voxel_types::kNotSerializable), 0)
      << "The voxel type of this layer is not serializable!";

  CHECK(!file_path.empty());
  std::fstream outfile;
  // Will APPEND to the current file in case outputting multiple layers on the
  // same file, depending on the flag.
  std::ios_base::openmode file_flags = std::fstream::out | std::fstream::binary;
  if (!clear_file) {
    file_flags |= std::fstream::app | std::fstream::ate;
  } else {
    file_flags |= std::fstream::trunc;
  }
  outfile.open(file_path, file_flags);
  if (!outfile.is_open()) {
    LOG(ERROR) << "Could not open file for writing: " << file_path;
    return false;
  }

  // Only serialize the blocks if there are any.
  // Count the number of blocks that need to be serialized.
  size_t num_blocks_to_write = 0u;
  if ((include_all_blocks && !block_map_.empty()) ||
      !blocks_to_include.empty()) {
    for (const BlockMapPair& pair : block_map_) {
      bool write_block_to_file = include_all_blocks;

      if (!write_block_to_file) {
        BlockIndexList::const_iterator it = std::find(
            blocks_to_include.begin(), blocks_to_include.end(), pair.first);
        if (it != blocks_to_include.end()) {
          ++num_blocks_to_write;
        }
      } else {
        ++num_blocks_to_write;
```

(continues on next page)

(continued from previous page)

```
      }
    }
  }
  if (include_all_blocks) {
    CHECK_EQ(num_blocks_to_write, block_map_.size());
  } else {
    CHECK_LE(num_blocks_to_write, block_map_.size());
    CHECK_LE(num_blocks_to_write, blocks_to_include.size());
  }

  // Write the total number of messages to the beginning of this file.
  // One layer header and then all the block maps
  const uint32_t num_messages = 1u + num_blocks_to_write;
  if (!utils::writeProtoMsgCountToStream(num_messages, &outfile)) {
    LOG(ERROR) << "Could not write message number to file.";
    outfile.close();
    return false;
  }

  // Write out the layer header.
  LayerProto proto_layer;
  getProto(&proto_layer);
  if (!utils::writeProtoMsgToStream(proto_layer, &outfile)) {
    LOG(ERROR) << "Could not write layer header message.";
    outfile.close();
    return false;
  }

  // Serialize blocks.
  for (const BlockMapPair& pair : block_map_) {
    bool write_block_to_file = include_all_blocks;
    if (!write_block_to_file) {
      BlockIndexList::const_iterator it = std::find(
          blocks_to_include.begin(), blocks_to_include.end(), pair.first);
      if (it != blocks_to_include.end()) {
        write_block_to_file = true;
      }
    }
    if (write_block_to_file) {
      BlockProto block_proto;
      pair.second->getProto(&block_proto);

      if (!utils::writeProtoMsgToStream(block_proto, &outfile)) {
        LOG(ERROR) << "Could not write block message.";
        outfile.close();
        return false;
      }
    }
  }
  outfile.close();
  return true;
}

template <typename VoxelType>
bool Layer<VoxelType>::addBlockFromProto(const BlockProto& block_proto,
                                         BlockMergingStrategy strategy) {
  CHECK_NE(getType().compare(voxel_types::kNotSerializable), 0)
```

(continues on next page)

(continued from previous page)

```
      << "The voxel type of this layer is not serializable!";

  if (isCompatible(block_proto)) {
    typename BlockType::Ptr block_ptr(new BlockType(block_proto));
    const BlockIndex block_index = getGridIndexFromOriginPoint<BlockIndex>(
        block_ptr->origin(), block_size_inv_);
    switch (strategy) {
      case BlockMergingStrategy::kProhibit:
        CHECK_EQ(block_map_.count(block_index), 0u)
            << "Block collision at index: " << block_index;
        block_map_[block_index] = block_ptr;
        break;
      case BlockMergingStrategy::kReplace:
        block_map_[block_index] = block_ptr;
        break;
      case BlockMergingStrategy::kDiscard:
        block_map_.insert(std::make_pair(block_index, block_ptr));
        break;
      case BlockMergingStrategy::kMerge: {
        typename BlockHashMap::iterator it = block_map_.find(block_index);
        if (it == block_map_.end()) {
          block_map_[block_index] = block_ptr;
        } else {
          it->second->mergeBlock(*block_ptr);
        }
      } break;
      default:
        LOG(FATAL) << "Unknown BlockMergingStrategy: "
                   << static_cast<int>(strategy);
        return false;
    }
    // Mark that this block has been updated.
    block_map_[block_index]->updated() = true;
  } else {
    LOG(ERROR)
        << "The blocks from this protobuf are not compatible with this layer!";
    return false;
  }
  return true;
}

template <typename VoxelType>
bool Layer<VoxelType>::isCompatible(const LayerProto& layer_proto) const {
  bool compatible = true;
  compatible &= (std::fabs(layer_proto.voxel_size() - voxel_size_) <
                 std::numeric_limits<FloatingPoint>::epsilon());
  compatible &= (layer_proto.voxels_per_side() == voxels_per_side_);
  compatible &= (getType().compare(layer_proto.type()) == 0);

  if (!compatible) {
    LOG(WARNING)
        << "Voxel size of the loaded map is: " << layer_proto.voxel_size()
        << " but the current map is: " << voxel_size_ << " check passed? "
        << (std::fabs(layer_proto.voxel_size() - voxel_size_) <
            std::numeric_limits<FloatingPoint>::epsilon())
        << "\nVPS of the loaded map is: " << layer_proto.voxels_per_side()
        << " but the current map is: " << voxels_per_side_ << " check passed? "
```

(continues on next page)

(continued from previous page)

```
        << (layer_proto.voxels_per_side() == voxels_per_side_)
        << "\nLayer type of the loaded map is: " << getType()
        << " but the current map is: " << layer_proto.type()
        << " check passed? " << (getType().compare(layer_proto.type()) == 0)
        << "\nAre the maps using the same floating-point type? "
        << (layer_proto.voxel_size() == voxel_size_) << std::endl;
  }
  return compatible;
}

template <typename VoxelType>
bool Layer<VoxelType>::isCompatible(const BlockProto& block_proto) const {
  bool compatible = true;
  compatible &= (std::fabs(block_proto.voxel_size() - voxel_size_) <
                 std::numeric_limits<FloatingPoint>::epsilon());
  compatible &=
      (block_proto.voxels_per_side() == static_cast<int>(voxels_per_side_));
  return compatible;
}

template <typename VoxelType>
size_t Layer<VoxelType>::getMemorySize() const {
  size_t size = 0u;

  // Calculate size of members
  size += sizeof(voxel_size_);
  size += sizeof(voxels_per_side_);
  size += sizeof(block_size_);
  size += sizeof(block_size_inv_);

  // Calculate size of blocks
  size_t num_blocks = getNumberOfAllocatedBlocks();
  if (num_blocks > 0u) {
    typename Block<VoxelType>::Ptr block = block_map_.begin()->second;
    size += num_blocks * block->getMemorySize();
  }
  return size;
}

template <typename VoxelType>
std::string Layer<VoxelType>::getType() const {
  return getVoxelType<VoxelType>();
}

}  // namespace voxblox

#endif  // VOXBLOX_CORE_LAYER_INL_H_
```

Includes

- ./Block.pb.h
- ./Layer.pb.h
- fstream
- glog/logging.h

- google/protobuf/io/coded_stream.h
- google/protobuf/io/zero_copy_stream.h
- google/protobuf/io/zero_copy_stream_impl.h
- google/protobuf/message.h
- google/protobuf/message_lite.h
- limits
- string
- utility
- voxblox/core/block.h (*File block.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/utils/protobuf_utils.h (*File protobuf_utils.h*)

Included By

- *File layer.h*

Namespaces

- *Namespace voxblox*

File layer_io.h

Contents

- *Definition (voxblox/include/voxblox/io/layer_io.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/io/layer_io.h)

Program Listing for File layer_io.h

Return to documentation for file (voxblox/include/voxblox/io/layer_io.h)

```
#ifndef VOXBLOX_IO_LAYER_IO_H_
#define VOXBLOX_IO_LAYER_IO_H_

#include <string>

#include <glog/logging.h>
```

(continues on next page)

(continued from previous page)

```
#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"

namespace voxblox {
namespace io {

template <typename VoxelType>
bool LoadBlocksFromFile(
    const std::string& file_path,
    typename Layer<VoxelType>::BlockMergingStrategy strategy,
    Layer<VoxelType>* layer_ptr);

template <typename VoxelType>
bool LoadBlocksFromFile(
    const std::string& file_path,
    typename Layer<VoxelType>::BlockMergingStrategy strategy,
    bool multiple_layer_support, Layer<VoxelType>* layer_ptr);

template <typename VoxelType>
bool LoadLayer(const std::string& file_path,
               typename Layer<VoxelType>::Ptr* layer_ptr);

template <typename VoxelType>
bool LoadLayer(const std::string& file_path, const bool multiple_layer_support,
               typename Layer<VoxelType>::Ptr* layer_ptr);

template <typename VoxelType>
bool SaveLayer(const Layer<VoxelType>& layer, const std::string& file_path,
               bool clear_file = true);

template <typename VoxelType>
bool SaveLayerSubset(const Layer<VoxelType>& layer,
                     const std::string& file_path,
                     const BlockIndexList& blocks_to_include,
                     bool include_all_blocks);

}  // namespace io
}  // namespace voxblox

#include "voxblox/io/layer_io_inl.h"

#endif  // VOXBLOX_IO_LAYER_IO_H_
```

Includes

- glog/logging.h
- string
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/io/layer_io_inl.h (*File layer_io_inl.h*)

Included By

- *File esdf_map.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::io*

File layer_io_inl.h

Contents

- *Definition (voxblox/include/voxblox/io/layer_io_inl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/io/layer_io_inl.h)

Program Listing for File layer_io_inl.h

Return to documentation for file (voxblox/include/voxblox/io/layer_io_inl.h)

```
#ifndef VOXBLOX_CORE_IO_LAYER_IO_INL_H_
#define VOXBLOX_CORE_IO_LAYER_IO_INL_H_

#include <fstream>

#include "./Block.pb.h"
#include "./Layer.pb.h"

#include "voxblox/utils/protobuf_utils.h"

namespace voxblox {
namespace io {

template <typename VoxelType>
bool LoadBlocksFromFile(
    const std::string& file_path,
    typename Layer<VoxelType>::BlockMergingStrategy strategy,
    bool multiple_layer_support, Layer<VoxelType>* layer_ptr) {
  CHECK_NOTNULL(layer_ptr);
  CHECK(!file_path.empty());

  // Open and check the file
```

(continues on next page)

(continued from previous page)

```
  std::fstream proto_file;
  proto_file.open(file_path, std::fstream::in);
  if (!proto_file.is_open()) {
    LOG(ERROR) << "Could not open protobuf file to load layer: " << file_path;
    return false;
  }

  // Byte offset result, used to keep track where we are in the file if
  // necessary.
  uint32_t tmp_byte_offset = 0;

  bool layer_found = false;

  do {
    // Get number of messages
    uint32_t num_protos;
    if (!utils::readProtoMsgCountToStream(&proto_file, &num_protos,
                                          &tmp_byte_offset)) {
      LOG(ERROR) << "Could not read number of messages.";
      return false;
    }

    if (num_protos == 0u) {
      LOG(WARNING) << "Empty protobuf file!";
      return false;
    }

    // Get header and check if it is compatible with existing layer.
    LayerProto layer_proto;
    if (!utils::readProtoMsgFromStream(&proto_file, &layer_proto,
                                       &tmp_byte_offset)) {
      LOG(ERROR) << "Could not read layer protobuf message.";
      return false;
    }

    if (layer_ptr->isCompatible(layer_proto)) {
      layer_found = true;
    } else if (!multiple_layer_support) {
      LOG(ERROR)
          << "The layer information read from file is not compatible with "
             "the current layer!";
      return false;
    } else {
      // Figure out how much offset to jump forward by. This is the number of
      // blocks * the block size... Actually maybe we just read them in? This
      // is probably easiest. Then if there's corrupted blocks, we abort.
      // We just don't add these to the layer.
      const size_t num_blocks = num_protos - 1;
      for (uint32_t block_idx = 0u; block_idx < num_blocks; ++block_idx) {
        BlockProto block_proto;
        if (!utils::readProtoMsgFromStream(&proto_file, &block_proto,
                                           &tmp_byte_offset)) {
          LOG(ERROR) << "Could not read block protobuf message number "
                     << block_idx;
          return false;
        }
      }
```

(continues on next page)

(continued from previous page)

```
      continue;
    }

    // Read all blocks and add them to the layer.
    const size_t num_blocks = num_protos - 1;
    for (uint32_t block_idx = 0u; block_idx < num_blocks; ++block_idx) {
      BlockProto block_proto;
      if (!utils::readProtoMsgFromStream(&proto_file, &block_proto,
                                         &tmp_byte_offset)) {
        LOG(ERROR) << "Could not read block protobuf message number "
                   << block_idx;
        return false;
      }

      if (!layer_ptr->addBlockFromProto(block_proto, strategy)) {
        LOG(ERROR) << "Could not add the block protobuf message to the layer!";
        return false;
      }
    }
  } while (multiple_layer_support && !layer_found && !proto_file.eof());
  return layer_found;
}

template <typename VoxelType>
bool LoadBlocksFromFile(
    const std::string& file_path,
    typename Layer<VoxelType>::BlockMergingStrategy strategy,
    Layer<VoxelType>* layer_ptr) {
  constexpr bool multiple_layer_support = false;
  return LoadBlocksFromFile(file_path, strategy, multiple_layer_support,
                            layer_ptr);
}

template <typename VoxelType>
bool LoadLayer(const std::string& file_path, const bool multiple_layer_support,
               typename Layer<VoxelType>::Ptr* layer_ptr) {
  CHECK_NOTNULL(layer_ptr);
  CHECK(!file_path.empty());

  // Open and check the file
  std::fstream proto_file;
  proto_file.open(file_path, std::fstream::in);
  if (!proto_file.is_open()) {
    LOG(ERROR) << "Could not open protobuf file to load layer: " << file_path;
    return false;
  }

  // Byte offset result, used to keep track where we are in the file if
  // necessary.
  uint32_t tmp_byte_offset = 0;

  bool layer_found = false;

  do {
    // Get number of messages
    uint32_t num_protos;
    if (!utils::readProtoMsgCountToStream(&proto_file, &num_protos,
```

(continues on next page)

(continued from previous page)

```
                                          &tmp_byte_offset)) {
    LOG(ERROR) << "Could not read number of messages.";
    return false;
  }

  if (num_protos == 0u) {
    LOG(WARNING) << "Empty protobuf file!";
    return false;
  }

  // Get header and check if it is compatible with existing layer.
  LayerProto layer_proto;
  if (!utils::readProtoMsgFromStream(&proto_file, &layer_proto,
                                     &tmp_byte_offset)) {
    LOG(ERROR) << "Could not read layer protobuf message.";
    return false;
  }

  if ((layer_proto.voxel_size() <= 0.0f) ||
      (layer_proto.voxels_per_side() == 0u)) {
    LOG(ERROR)
        << "Invalid parameter in layer protobuf message. Check the format.";
    return false;
  }

  if (getVoxelType<VoxelType>().compare(layer_proto.type()) == 0) {
    layer_found = true;
  } else if (!multiple_layer_support) {
    LOG(ERROR)
        << "The layer information read from file is not compatible with "
           "the current layer!";
    return false;
  } else {
    // Figure out how much offset to jump forward by. This is the number of
    // blocks * the block size... Actually maybe we just read them in? This
    // is probably easiest. Then if there's corrupted blocks, we abort.
    // We just don't add these to the layer.
    const size_t num_blocks = num_protos - 1;
    for (uint32_t block_idx = 0u; block_idx < num_blocks; ++block_idx) {
      BlockProto block_proto;
      if (!utils::readProtoMsgFromStream(&proto_file, &block_proto,
                                         &tmp_byte_offset)) {
        LOG(ERROR) << "Could not read block protobuf message number "
                   << block_idx;
        return false;
      }
    }
    continue;
  }

  *layer_ptr = aligned_shared<Layer<VoxelType> >(layer_proto);
  CHECK(*layer_ptr);

  // Read all blocks and add them to the layer.
  const size_t num_blocks = num_protos - 1;
  for (uint32_t block_idx = 0u; block_idx < num_blocks; ++block_idx) {
    BlockProto block_proto;
```

(continues on next page)

(continued from previous page)

```
      if (!utils::readProtoMsgFromStream(&proto_file, &block_proto,
                                         &tmp_byte_offset)) {
        LOG(ERROR) << "Could not read block protobuf message number "
                   << block_idx;
        return false;
      }

      if (!(*layer_ptr)
               ->addBlockFromProto(
                   block_proto,
                   Layer<VoxelType>::BlockMergingStrategy::kProhibit)) {
        LOG(ERROR) << "Could not add the block protobuf message to the layer!";
        return false;
      }
    }
  } while (multiple_layer_support && !layer_found && !proto_file.eof());
  return layer_found;
}

// By default loads without multiple layer support (i.e., only checks the first
// layer in the file).
template <typename VoxelType>
bool LoadLayer(const std::string& file_path,
               typename Layer<VoxelType>::Ptr* layer_ptr) {
  constexpr bool multiple_layer_support = false;
  return LoadLayer<VoxelType>(file_path, multiple_layer_support, layer_ptr);
}

template <typename VoxelType>
bool SaveLayer(const Layer<VoxelType>& layer, const std::string& file_path,
               bool clear_file) {
  CHECK(!file_path.empty());
  return layer.saveToFile(file_path, clear_file);
}

template <typename VoxelType>
bool SaveLayerSubset(const Layer<VoxelType>& layer,
                     const std::string& file_path,
                     const BlockIndexList& blocks_to_include,
                     bool include_all_blocks) {
  CHECK(!file_path.empty());
  return layer.saveSubsetToFile(file_path, blocks_to_include,
                                include_all_blocks);
}

}  // namespace io
}  // namespace voxblox

#endif  // VOXBLOX_CORE_IO_LAYER_IO_INL_H_
```

Includes

- ./Block.pb.h
- ./Layer.pb.h

- fstream
- voxblox/utils/protobuf_utils.h (*File protobuf_utils.h*)

Included By

- *File layer_io.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::io*

File layer_test_utils.h

Contents

- *Definition (voxblox/include/voxblox/test/layer_test_utils.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/test/layer_test_utils.h)

Program Listing for File layer_test_utils.h

Return to documentation for file (voxblox/include/voxblox/test/layer_test_utils.h)

```
#ifndef VOXBLOX_TEST_LAYER_TEST_UTILS_H_
#define VOXBLOX_TEST_LAYER_TEST_UTILS_H_

#include <gtest/gtest.h>

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
namespace test {

template <typename VoxelType>
class LayerTest {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  void CompareLayers(const Layer<VoxelType>& layer_A,
                     const Layer<VoxelType>& layer_B) const {
    EXPECT_NEAR(layer_A.voxel_size(), layer_B.voxel_size(), kTolerance);
    EXPECT_NEAR(layer_A.block_size(), layer_B.block_size(), kTolerance);
```

(continues on next page)

(continued from previous page)

```
  EXPECT_EQ(layer_A.voxels_per_side(), layer_B.voxels_per_side());
  EXPECT_EQ(layer_A.getNumberOfAllocatedBlocks(),
            layer_B.getNumberOfAllocatedBlocks());

  BlockIndexList blocks_A, blocks_B;
  layer_A.getAllAllocatedBlocks(&blocks_A);
  layer_B.getAllAllocatedBlocks(&blocks_B);
  EXPECT_EQ(blocks_A.size(), blocks_B.size());
  for (const BlockIndex& index_A : blocks_A) {
    BlockIndexList::const_iterator it =
        std::find(blocks_B.begin(), blocks_B.end(), index_A);
    if (it != blocks_B.end()) {
      const Block<VoxelType>& block_A = layer_A.getBlockByIndex(index_A);
      const Block<VoxelType>& block_B = layer_B.getBlockByIndex(*it);
      CompareBlocks(block_A, block_B);
    } else {
      ADD_FAILURE();
      LOG(ERROR) << "Block at index [" << index_A.transpose()
                 << "] in layer_A does not exists in layer_B";
    }
  }
  for (const BlockIndex& index_B : blocks_B) {
    BlockIndexList::const_iterator it =
        std::find(blocks_A.begin(), blocks_A.end(), index_B);
    if (it != blocks_A.end()) {
      const Block<VoxelType>& block_B = layer_A.getBlockByIndex(index_B);
      const Block<VoxelType>& block_A = layer_B.getBlockByIndex(*it);
      CompareBlocks(block_B, block_A);
    } else {
      ADD_FAILURE();
      LOG(ERROR) << "Block at index [" << index_B.transpose()
                 << "] in layer_B does not exists in layer_A";
    }
  }

  EXPECT_EQ(layer_A.getMemorySize(), layer_B.getMemorySize());
}

void CompareBlocks(const Block<VoxelType>& block_A,
                   const Block<VoxelType>& block_B) const {
  EXPECT_NEAR(block_A.voxel_size(), block_B.voxel_size(), kTolerance);
  EXPECT_NEAR(block_A.block_size(), block_B.block_size(), kTolerance);
  EXPECT_EQ(block_A.voxels_per_side(), block_B.voxels_per_side());

  EXPECT_NEAR(block_A.origin().x(), block_B.origin().x(), kTolerance);
  EXPECT_NEAR(block_A.origin().y(), block_B.origin().y(), kTolerance);
  EXPECT_NEAR(block_A.origin().z(), block_B.origin().z(), kTolerance);

  EXPECT_EQ(block_A.num_voxels(), block_B.num_voxels());
  for (size_t voxel_idx = 0u; voxel_idx < block_A.num_voxels(); ++voxel_idx) {
    CompareVoxel(block_A.getVoxelByLinearIndex(voxel_idx),
                 block_B.getVoxelByLinearIndex(voxel_idx));
  }
}

void CompareVoxel(const VoxelType& voxel_A, const VoxelType& voxel_B) const;
```

(continues on next page)

(continued from previous page)

```
  static constexpr double kTolerance = 1e-10;
};

template <typename VoxelType>
void LayerTest<VoxelType>::CompareVoxel(const VoxelType& /* voxel_A */,
                                        const VoxelType& /* voxel_B */) const {
  LOG(FATAL) << "Not implemented for this voxel type!";
}

template <>
void LayerTest<EsdfVoxel>::CompareVoxel(const EsdfVoxel& voxel_A,
                                        const EsdfVoxel& voxel_B) const {
  constexpr double kTolerance = 1e-10;

  EXPECT_NEAR(voxel_A.distance, voxel_B.distance, kTolerance);

  // Flags
  EXPECT_EQ(voxel_A.observed, voxel_B.observed);
  EXPECT_EQ(voxel_A.hallucinated, voxel_B.hallucinated);
  EXPECT_EQ(voxel_A.in_queue, voxel_B.in_queue);
  EXPECT_EQ(voxel_A.fixed, voxel_B.fixed);

  EXPECT_EQ(voxel_A.parent.x(), voxel_B.parent.x());
  EXPECT_EQ(voxel_A.parent.y(), voxel_B.parent.y());
  EXPECT_EQ(voxel_A.parent.z(), voxel_B.parent.z());
}

template <>
void LayerTest<OccupancyVoxel>::CompareVoxel(
    const OccupancyVoxel& voxel_A, const OccupancyVoxel& voxel_B) const {
  constexpr double kTolerance = 1e-10;

  EXPECT_NEAR(voxel_A.probability_log, voxel_B.probability_log, kTolerance);
  EXPECT_EQ(voxel_A.observed, voxel_B.observed);
}

template <>
void LayerTest<TsdfVoxel>::CompareVoxel(const TsdfVoxel& voxel_A,
                                        const TsdfVoxel& voxel_B) const {
  EXPECT_NEAR(voxel_A.distance, voxel_B.distance, kTolerance);
  EXPECT_NEAR(voxel_A.weight, voxel_B.weight, kTolerance);
  EXPECT_EQ(voxel_A.color.r, voxel_B.color.r);
  EXPECT_EQ(voxel_A.color.g, voxel_B.color.g);
  EXPECT_EQ(voxel_A.color.b, voxel_B.color.b);
  EXPECT_EQ(voxel_A.color.a, voxel_B.color.a);
}

template <>
void LayerTest<IntensityVoxel>::CompareVoxel(
    const IntensityVoxel& voxel_A, const IntensityVoxel& voxel_B) const {
  EXPECT_NEAR(voxel_A.intensity, voxel_B.intensity, kTolerance);
  EXPECT_NEAR(voxel_A.weight, voxel_B.weight, kTolerance);
}

template <typename VoxelType>
void fillVoxelWithTestData(size_t x, size_t y, size_t z, VoxelType* voxel);
```

(continues on next page)

(continued from previous page)

```
template <typename VoxelType>
void SetUpTestLayer(const IndexElement block_volume_diameter,
                    const IndexElement block_volume_offset,
                    Layer<VoxelType>* layer) {
  CHECK_NOTNULL(layer);

  const IndexElement half_idx_range = block_volume_diameter / 2;

  // For better readability.
  const IndexElement& min_idx = -half_idx_range + block_volume_offset;
  const IndexElement& max_idx = half_idx_range + block_volume_offset;

  for (IndexElement x = min_idx; x <= max_idx; ++x) {
    for (IndexElement y = min_idx; y <= max_idx; ++y) {
      for (IndexElement z = min_idx; z <= max_idx; ++z) {
        BlockIndex block_idx = {x, y, z};
        typename Block<VoxelType>::Ptr block =
            layer->allocateBlockPtrByIndex(block_idx);
        VoxelType& voxel = block->getVoxelByLinearIndex(
            (x * z + y) % layer->voxels_per_side());

        fillVoxelWithTestData(x, y, z, &voxel);

        block->has_data() = true;
      }
    }
  }
  const double size_in_MB = static_cast<double>(layer->getMemorySize()) * 1e-6;
  std::cout << std::endl
            << "Set up a test layer of size " << size_in_MB << " MB";
}

template <typename VoxelType>
void SetUpTestLayer(const IndexElement block_volume_diameter,
                    Layer<VoxelType>* layer) {
  constexpr IndexElement kBlockVolumeOffset = 0u;
  SetUpTestLayer(block_volume_diameter, kBlockVolumeOffset, layer);
}

template <>
inline void fillVoxelWithTestData(size_t x, size_t y, size_t z,
                                  TsdfVoxel* voxel) {
  CHECK_NOTNULL(voxel);
  voxel->distance = x * y * 0.66 + z;
  voxel->weight = y * z * 0.33 + x;
  voxel->color.r = static_cast<uint8_t>(x % 255);
  voxel->color.g = static_cast<uint8_t>(y % 255);
  voxel->color.b = static_cast<uint8_t>(z % 255);
  voxel->color.a = static_cast<uint8_t>(x + y % 255);
}

template <>
inline void fillVoxelWithTestData(size_t x, size_t y, size_t z,
                                  EsdfVoxel* voxel) {
  CHECK_NOTNULL(voxel);
  voxel->distance = x * y * 0.66 + z;
  voxel->parent.x() = x % INT8_MAX;
```

(continues on next page)

(continued from previous page)

```
  voxel->parent.y() = y % INT8_MAX;
  voxel->parent.z() = z % INT8_MAX;

  voxel->observed = true;
  voxel->in_queue = true;
  voxel->fixed = true;
}

template <>
inline void fillVoxelWithTestData(size_t x, size_t y, size_t z,
                                  OccupancyVoxel* voxel) {
  CHECK_NOTNULL(voxel);
  voxel->probability_log = x * y * 0.66 + z;
  voxel->observed = true;
}

template <>
inline void fillVoxelWithTestData(size_t x, size_t y, size_t z,
                                  IntensityVoxel* voxel) {
  CHECK_NOTNULL(voxel);
  voxel->intensity = x * y * 0.66 + z;
  voxel->weight = y * z * 0.33 + x;
}

}  // namespace test
}  // namespace voxblox

#endif  // VOXBLOX_TEST_LAYER_TEST_UTILS_H_
```

Includes

- gtest/gtest.h
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::test*

Classes

- *Template Class LayerTest*

File layer_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/layer_utils.h)*
- *Includes*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/layer_utils.h)

Program Listing for File layer_utils.h

Return to documentation for file (voxblox/include/voxblox/utils/layer_utils.h)

```
#ifndef VOXBLOX_UTILS_LAYER_UTILS_H_
#define VOXBLOX_UTILS_LAYER_UTILS_H_

#include <utility>

#include <Eigen/Core>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
namespace utils {

template <typename VoxelType>
bool isSameVoxel(const VoxelType& /* voxel_A */,
                 const VoxelType& /* voxel_B */) {
  LOG(FATAL) << "Not implemented for this voxel type!";
  return false;
}

template <typename VoxelType>
bool isSameBlock(const Block<VoxelType>& block_A,
                 const Block<VoxelType>& block_B) {
  bool is_the_same = true;

  constexpr double kTolerance = 1e-10;

  is_the_same &=
      std::abs(block_A.voxel_size() - block_B.voxel_size()) < kTolerance;
  is_the_same &=
      std::abs(block_A.block_size() - block_B.block_size()) < kTolerance;
  is_the_same &= block_A.voxels_per_side() == block_B.voxels_per_side();

  is_the_same &=
      std::abs(block_A.origin().x() - block_B.origin().x()) < kTolerance;
  is_the_same &=
      std::abs(block_A.origin().y() - block_B.origin().y()) < kTolerance;
  is_the_same &=
      std::abs(block_A.origin().z() - block_B.origin().z()) < kTolerance;
```

(continues on next page)

(continued from previous page)

```
  is_the_same &= block_A.num_voxels() == block_B.num_voxels();

  for (size_t voxel_idx = 0u; voxel_idx < block_A.num_voxels(); ++voxel_idx) {
    is_the_same &= isSameVoxel(block_A.getVoxelByLinearIndex(voxel_idx),
                               block_B.getVoxelByLinearIndex(voxel_idx));
  }
  return is_the_same;
}

template <typename VoxelType>
bool isSameLayer(const Layer<VoxelType>& layer_A,
                 const Layer<VoxelType>& layer_B) {
  constexpr double kTolerance = 1e-10;

  bool is_the_same = true;

  is_the_same &=
      std::abs(layer_A.voxel_size() - layer_B.voxel_size()) < kTolerance;
  is_the_same &=
      std::abs(layer_A.block_size() - layer_B.block_size()) < kTolerance;

  is_the_same &= layer_A.voxels_per_side() == layer_B.voxels_per_side();
  is_the_same &= layer_A.getNumberOfAllocatedBlocks() ==
                 layer_B.getNumberOfAllocatedBlocks();

  BlockIndexList blocks_A, blocks_B;
  layer_A.getAllAllocatedBlocks(&blocks_A);
  layer_B.getAllAllocatedBlocks(&blocks_B);
  is_the_same &= blocks_A.size() == blocks_B.size();

  for (const BlockIndex& index_A : blocks_A) {
    const BlockIndexList::const_iterator it =
        std::find(blocks_B.begin(), blocks_B.end(), index_A);
    if (it != blocks_B.end()) {
      const Block<VoxelType>& block_A = layer_A.getBlockByIndex(index_A);
      const Block<VoxelType>& block_B = layer_B.getBlockByIndex(*it);
      bool is_same_block = isSameBlock(block_A, block_B);
      LOG_IF(ERROR, !is_same_block)
          << "Block at index [" << index_A.transpose()
          << "] in layer_A is not the same as in layer_B";
      is_the_same &= is_same_block;
    } else {
      LOG(ERROR) << "Block at index [" << index_A.transpose()
                 << "] in layer_A does not exists in layer_B";
      return false;
    }
  }
  for (const BlockIndex& index_B : blocks_B) {
    const BlockIndexList::const_iterator it =
        std::find(blocks_A.begin(), blocks_A.end(), index_B);
    if (it != blocks_A.end()) {
      const Block<VoxelType>& block_B = layer_A.getBlockByIndex(index_B);
      const Block<VoxelType>& block_A = layer_B.getBlockByIndex(*it);
      bool is_same_block = isSameBlock(block_B, block_A);
      LOG_IF(ERROR, !is_same_block)
          << "Block at index [" << index_B.transpose()
          << "] in layer_B is not the same as in layer_A";
```

(continues on next page)

(continued from previous page)

```
      is_the_same &= is_same_block;
    } else {
      LOG(ERROR) << "Block at index [" << index_B.transpose()
                 << "] in layer_B does not exists in layer_A";
      return false;
    }
  }

  is_the_same &= layer_A.getMemorySize() == layer_B.getMemorySize();
  return is_the_same;
}

template <>
bool isSameVoxel(const TsdfVoxel& voxel_A, const TsdfVoxel& voxel_B);

template <>
bool isSameVoxel(const EsdfVoxel& voxel_A, const EsdfVoxel& voxel_B);

template <>
bool isSameVoxel(const OccupancyVoxel& voxel_A, const OccupancyVoxel& voxel_B);

template <typename VoxelType>
void centerBlocksOfLayer(Layer<VoxelType>* layer, Point* new_layer_origin) {
  CHECK_NOTNULL(layer);
  CHECK_NOTNULL(new_layer_origin);

  // Compute the exact cenroid of all allocated block indices.
  Point centroid = Point::Zero();
  BlockIndexList block_indices;
  layer->getAllAllocatedBlocks(&block_indices);
  for (const BlockIndex block_index : block_indices) {
    centroid += layer->getBlockByIndex(block_index).origin();
  }
  centroid /= static_cast<FloatingPoint>(block_indices.size());

  // Round to nearest block index to centroid.
  centroid /= layer->block_size();
  const BlockIndex index_centroid =
      (centroid + 0.5 * Point::Ones()).cast<IndexElement>();

  // Return the new origin expressed in the old origins coordinate frame.
  const FloatingPoint block_size = layer->block_size();
  *new_layer_origin = index_centroid.cast<FloatingPoint>() * block_size;

  VLOG(3) << "The new origin of the coordinate frame (expressed in the old "
          << "coordinate frame) is: " << new_layer_origin->transpose();

  // Loop over all blocks and change their spatial indices.
  // The only way to do this is to remove them all, store them in a temporary
  // hash map and re-insert them again. This sounds worse than it is, the blocks
  // are all shared ptrs and therefore only a negligible amount of real memory
  // operations is necessary.
  const size_t num_allocated_blocks_before =
      layer->getNumberOfAllocatedBlocks();
  typename Layer<VoxelType>::BlockHashMap temporary_map;
  for (const BlockIndex& block_index : block_indices) {
    typename Block<VoxelType>::Ptr block_ptr =
```

(continues on next page)

(continued from previous page)

```
        layer->getBlockPtrByIndex(block_index);
    const Point new_origin = block_ptr->origin() - *new_layer_origin;
    block_ptr->setOrigin(new_origin);

    // Extract block and shift block index.
    temporary_map.emplace(block_index - index_centroid, block_ptr);
  }

  layer->removeAllBlocks();
  CHECK_EQ(layer->getNumberOfAllocatedBlocks(), 0u);

  // Insert into layer again.
  for (const std::pair<const BlockIndex, typename Block<VoxelType>::Ptr>&
           idx_block_pair : temporary_map) {
    layer->insertBlock(idx_block_pair);
  }
  CHECK_EQ(layer->getNumberOfAllocatedBlocks(), num_allocated_blocks_before);
}

}  // namespace utils
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_LAYER_UTILS_H_
```

Includes

- Eigen/Core
- utility
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::utils*

File marching_cubes.h

Contents

- *Definition (voxblox/include/voxblox/mesh/marching_cubes.h)*
- *Includes*
- *Included By*
- *Namespaces*

- *Classes*

Definition (voxblox/include/voxblox/mesh/marching_cubes.h)

Program Listing for File marching_cubes.h

Return to documentation for file (voxblox/include/voxblox/mesh/marching_cubes.h)

```
// The MIT License (MIT)
// Copyright (c) 2014 Matthew Klingensmith and Ivan Dryanovski
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef VOXBLOX_MESH_MARCHING_CUBES_H_
#define VOXBLOX_MESH_MARCHING_CUBES_H_

#include "voxblox/mesh/mesh.h"

namespace voxblox {

class MarchingCubes {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  static int kTriangleTable[256][16];
  static int kEdgeIndexPairs[12][2];

  MarchingCubes() {}
  virtual ~MarchingCubes() {}

  static void meshCube(
      const Eigen::Matrix<FloatingPoint, 3, 8>& vertex_coordinates,
      const Eigen::Matrix<FloatingPoint, 8, 1>& vertex_sdf,
      TriangleVector* triangles) {
    DCHECK(triangles != NULL);

    const int index = calculateVertexConfiguration(vertex_sdf);
```

(continues on next page)

(continued from previous page)

```
    Eigen::Matrix<FloatingPoint, 3, 12> edge_coords;
    interpolateEdgeVertices(vertex_coordinates, vertex_sdf, &edge_coords);

    const int* table_row = kTriangleTable[index];

    int edge_index = 0;
    int table_col = 0;
    while ((edge_index = table_row[table_col]) != -1) {
      Triangle triangle;
      triangle.col(0) = edge_coords.col(edge_index);
      edge_index = table_row[table_col + 1];
      triangle.col(1) = edge_coords.col(edge_index);
      edge_index = table_row[table_col + 2];
      triangle.col(2) = edge_coords.col(edge_index);
      triangles->push_back(triangle);
      table_col += 3;
    }
  }

  static void meshCube(const Eigen::Matrix<FloatingPoint, 3, 8>& vertex_coords,
                       const Eigen::Matrix<FloatingPoint, 8, 1>& vertex_sdf,
                       VertexIndex* next_index, Mesh* mesh) {
    DCHECK(next_index != NULL);
    DCHECK(mesh != NULL);
    const int index = calculateVertexConfiguration(vertex_sdf);

    // No edges in this cube.
    if (index == 0) {
      return;
    }

    Eigen::Matrix<FloatingPoint, 3, 12> edge_vertex_coordinates;
    interpolateEdgeVertices(vertex_coords, vertex_sdf,
                            &edge_vertex_coordinates);

    const int* table_row = kTriangleTable[index];

    int table_col = 0;
    while (table_row[table_col] != -1) {
      mesh->vertices.emplace_back(
          edge_vertex_coordinates.col(table_row[table_col + 2]));
      mesh->vertices.emplace_back(
          edge_vertex_coordinates.col(table_row[table_col + 1]));
      mesh->vertices.emplace_back(
          edge_vertex_coordinates.col(table_row[table_col]));
      mesh->indices.push_back(*next_index);
      mesh->indices.push_back((*next_index) + 1);
      mesh->indices.push_back((*next_index) + 2);
      const Point& p0 = mesh->vertices[*next_index];
      const Point& p1 = mesh->vertices[*next_index + 1];
      const Point& p2 = mesh->vertices[*next_index + 2];
      Point px = (p1 - p0);
      Point py = (p2 - p0);
      Point n = px.cross(py).normalized();
      mesh->normals.push_back(n);
      mesh->normals.push_back(n);
      mesh->normals.push_back(n);
```

(continues on next page)

(continued from previous page)

```
      *next_index += 3;
      table_col += 3;
    }
  }

  static int calculateVertexConfiguration(
      const Eigen::Matrix<FloatingPoint, 8, 1>& vertex_sdf) {
    return (vertex_sdf(0) < 0 ? (1 << 0) : 0) |
           (vertex_sdf(1) < 0 ? (1 << 1) : 0) |
           (vertex_sdf(2) < 0 ? (1 << 2) : 0) |
           (vertex_sdf(3) < 0 ? (1 << 3) : 0) |
           (vertex_sdf(4) < 0 ? (1 << 4) : 0) |
           (vertex_sdf(5) < 0 ? (1 << 5) : 0) |
           (vertex_sdf(6) < 0 ? (1 << 6) : 0) |
           (vertex_sdf(7) < 0 ? (1 << 7) : 0);
  }

  static void interpolateEdgeVertices(
      const Eigen::Matrix<FloatingPoint, 3, 8>& vertex_coords,
      const Eigen::Matrix<FloatingPoint, 8, 1>& vertex_sdf,
      Eigen::Matrix<FloatingPoint, 3, 12>* edge_coords) {
    DCHECK(edge_coords != NULL);
    for (std::size_t i = 0; i < 12; ++i) {
      const int* pairs = kEdgeIndexPairs[i];
      const int edge0 = pairs[0];
      const int edge1 = pairs[1];
      // Only interpolate along edges where there is a zero crossing.
      if ((vertex_sdf(edge0) < 0 && vertex_sdf(edge1) >= 0) ||
          (vertex_sdf(edge0) >= 0 && vertex_sdf(edge1) < 0))
        edge_coords->col(i) = interpolateVertex(
            vertex_coords.col(edge0), vertex_coords.col(edge1),
            vertex_sdf(edge0), vertex_sdf(edge1));
    }
  }

  static inline Point interpolateVertex(const Point& vertex1,
                                        const Point& vertex2, float sdf1,
                                        float sdf2) {
    static constexpr FloatingPoint kMinSdfDifference = 1e-6;
    const FloatingPoint sdf_diff = sdf1 - sdf2;
    // Only compute the actual interpolation value if the sdf_difference is not
    // too small, this is to counteract issues with floating point precision.
    if (std::abs(sdf_diff) >= kMinSdfDifference) {
      const FloatingPoint t = sdf1 / sdf_diff;
      return Point(vertex1 + t * (vertex2 - vertex1));
    } else {
      return Point(0.5 * (vertex1 + vertex2));
    }
  }
};

}  // namespace voxblox

#endif  // VOXBLOX_MESH_MARCHING_CUBES_H_
```

Includes

- voxblox/mesh/mesh.h (*File mesh.h*)

Included By

- *File mesh_integrator.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class MarchingCubes*

File merge_integration.h

Contents

- *Definition (voxblox/include/voxblox/integrator/merge_integration.h)*
- *Includes*
- *Namespaces*

Definition (voxblox/include/voxblox/integrator/merge_integration.h)

Program Listing for File merge_integration.h

Return to documentation for file (voxblox/include/voxblox/integrator/merge_integration.h)

```
#ifndef VOXBLOX_INTEGRATOR_MERGE_INTEGRATION_H_
#define VOXBLOX_INTEGRATOR_MERGE_INTEGRATION_H_

#include <algorithm>
#include <utility>
#include <vector>

#include <glog/logging.h>

#include "voxblox/interpolator/interpolator.h"

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
```

(continues on next page)

(continued from previous page)

```
static const FloatingPoint kUnitCubeDiagonalLength = std::sqrt(3.0);

template <typename VoxelType>
void mergeLayerAintoLayerB(const Layer<VoxelType>& layer_A,
                           Layer<VoxelType>* layer_B) {
  CHECK_NOTNULL(layer_B);
  // if voxel layout is different resample layer A to match B
  const Layer<VoxelType>* layer_A_ptr;
  Layer<VoxelType> layer_A_resampled(layer_B->voxel_size(),
                                     layer_B->voxels_per_side());

  if ((layer_A.voxel_size() != layer_B->voxel_size()) ||
      (layer_A.voxels_per_side() != layer_B->voxels_per_side())) {
    resampleLayer(layer_A, &layer_A_resampled);
    layer_A_ptr = &layer_A_resampled;
  } else {
    layer_A_ptr = &layer_A;
  }

  BlockIndexList block_idx_list_A;
  layer_A.getAllAllocatedBlocks(&block_idx_list_A);

  for (const BlockIndex& block_idx : block_idx_list_A) {
    typename Block<VoxelType>::ConstPtr block_A_ptr =
        layer_A_ptr->getBlockPtrByIndex(block_idx);
    typename Block<VoxelType>::Ptr block_B_ptr =
        layer_B->getBlockPtrByIndex(block_idx);

    if (!block_B_ptr) {
      block_B_ptr = layer_B->allocateBlockPtrByIndex(block_idx);
    }

    if ((block_A_ptr != nullptr) && (block_B_ptr != nullptr)) {
      block_B_ptr->mergeBlock(*block_A_ptr);
    }
  }
}

template <typename VoxelType>
void mergeLayerAintoLayerB(const Layer<VoxelType>& layer_A,
                           const Transformation& T_B_A,
                           Layer<VoxelType>* layer_B,
                           bool use_naive_method = false) {
  Layer<VoxelType> layer_A_transformed(layer_B->voxel_size(),
                                       layer_B->voxels_per_side());

  if (use_naive_method) {
    naiveTransformLayer(layer_A, T_B_A, &layer_A_transformed);
  } else {
    transformLayer(layer_A, T_B_A, &layer_A_transformed);
  }

  mergeLayerAintoLayerB(layer_A_transformed, layer_B);
}

template <typename VoxelType>
```

(continues on next page)

(continued from previous page)

```
void resampleLayer(const Layer<VoxelType>& layer_in,
                   Layer<VoxelType>* layer_out) {
  CHECK_NOTNULL(layer_out);
  transformLayer(layer_in, Transformation(), layer_out);
}

template <typename VoxelType>
void naiveTransformLayer(const Layer<VoxelType>& layer_in,
                         const Transformation& T_out_in,
                         Layer<VoxelType>* layer_out) {
  BlockIndexList block_idx_list_in;
  layer_in.getAllAllocatedBlocks(&block_idx_list_in);

  Interpolator<VoxelType> interpolator(&layer_in);

  for (const BlockIndex& block_idx : block_idx_list_in) {
    const Block<VoxelType>& input_block = layer_in.getBlockByIndex(block_idx);

    for (IndexElement input_linear_voxel_idx = 0;
         input_linear_voxel_idx <
         static_cast<IndexElement>(input_block.num_voxels());
         ++input_linear_voxel_idx) {
      // find voxel centers location in the output
      const Point voxel_center =
          T_out_in *
          input_block.computeCoordinatesFromLinearIndex(input_linear_voxel_idx);

      const GlobalIndex global_output_voxel_idx =
          getGridIndexFromPoint<GlobalIndex>(voxel_center,
                                             layer_out->voxel_size_inv());

      // allocate it in the output
      typename Block<VoxelType>::Ptr output_block =
          layer_out->allocateBlockPtrByIndex(getBlockIndexFromGlobalVoxelIndex(
              global_output_voxel_idx, layer_out->voxels_per_side_inv()));

      if (output_block == nullptr) {
        std::cerr << "invalid block" << std::endl;
      }

      // get the output voxel
      VoxelType& output_voxel =
          output_block->getVoxelByVoxelIndex(getLocalFromGlobalVoxelIndex(
              global_output_voxel_idx, layer_out->voxels_per_side()));

      if (interpolator.getVoxel(voxel_center, &output_voxel, false)) {
        output_block->has_data() = true;
      }
    }
  }
}

template <typename VoxelType>
void transformLayer(const Layer<VoxelType>& layer_in,
                    const Transformation& T_out_in,
                    Layer<VoxelType>* layer_out) {
  CHECK_NOTNULL(layer_out);
```

(continues on next page)

(continued from previous page)

```
// first mark all the blocks in the output layer that may be filled by the
// input layer (we are conservative here approximating the input blocks as
// spheres of diameter sqrt(3)*block_size)
IndexSet block_idx_set;

BlockIndexList block_idx_list_in;
layer_in.getAllAllocatedBlocks(&block_idx_list_in);

for (const BlockIndex& block_idx : block_idx_list_in) {
  const Point c_in =
      getCenterPointFromGridIndex(block_idx, layer_in.block_size());

  // forwards transform of center
  const Point c_out = T_out_in * c_in;

  // Furthest center point of neighboring blocks.
  FloatingPoint offset =
      kUnitCubeDiagonalLength * layer_in.block_size() * 0.5;

  // Add index of all blocks in range to set.
  for (FloatingPoint x = c_out.x() - offset; x < c_out.x() + offset;
       x += layer_out->block_size()) {
    for (FloatingPoint y = c_out.y() - offset; y < c_out.y() + offset;
         y += layer_out->block_size()) {
      for (FloatingPoint z = c_out.z() - offset; z < c_out.z() + offset;
           z += layer_out->block_size()) {
        const Point current_center_out = Point(x, y, z);
        BlockIndex current_idx = getGridIndexFromPoint<BlockIndex>(
            current_center_out, 1.0f / layer_out->block_size());
        block_idx_set.insert(current_idx);
      }
    }
  }
}

// get inverse transform
const Transformation T_in_out = T_out_in.inverse();

Interpolator<VoxelType> interpolator(&layer_in);

// we now go through all the blocks in the output layer and interpolate the
// input layer at the center of each output voxel position
for (const BlockIndex& block_idx : block_idx_set) {
  typename Block<VoxelType>::Ptr block =
      layer_out->allocateBlockPtrByIndex(block_idx);

  for (IndexElement voxel_idx = 0;
       voxel_idx < static_cast<IndexElement>(block->num_voxels());
       ++voxel_idx) {
    VoxelType& voxel = block->getVoxelByLinearIndex(voxel_idx);

    // find voxel centers location in the input
    const Point voxel_center =
        T_in_out * block->computeCoordinatesFromLinearIndex(voxel_idx);

    // interpolate voxel
```

(continues on next page)

(continued from previous page)

```
      if (interpolator.getVoxel(voxel_center, &voxel, true)) {
        block->has_data() = true;

        // if interpolated value fails use nearest
      } else if (interpolator.getVoxel(voxel_center, &voxel, false)) {
        block->has_data() = true;
      }
    }

    if (!block->has_data()) {
      layer_out->removeBlock(block_idx);
    }
  }
}

typedef std::pair<voxblox::Layer<voxblox::TsdfVoxel>::Ptr,
                  voxblox::Layer<voxblox::TsdfVoxel>::Ptr>
    AlignedLayerAndErrorLayer;
typedef std::vector<AlignedLayerAndErrorLayer> AlignedLayerAndErrorLayers;

template <typename VoxelType>
void evaluateLayerRmseAtPoses(
    const utils::VoxelEvaluationMode& voxel_evaluation_mode,
    const Layer<VoxelType>& layer_A, const Layer<VoxelType>& layer_B,
    const std::vector<Transformation>& transforms_A_B,
    std::vector<utils::VoxelEvaluationDetails>* voxel_evaluation_details_vector,
    std::vector<std::pair<typename voxblox::Layer<VoxelType>::Ptr,
                          typename voxblox::Layer<VoxelType>::Ptr>>*
        aligned_layers_and_error_layers = nullptr) {
  CHECK_NOTNULL(voxel_evaluation_details_vector);

  // Check if layers are compatible.
  CHECK_NEAR(layer_A.voxel_size(), layer_B.voxel_size(), 1e-8);
  CHECK_EQ(layer_A.voxels_per_side(), layer_B.voxels_per_side());

  // Check if world TSDF layer agrees with merged object at all object poses.

  for (size_t i = 0u; i < transforms_A_B.size(); ++i) {
    const Transformation& transform_A_B = transforms_A_B[i];

    // Layer B transformed to the coordinate frame A.
    typename Layer<VoxelType>::Ptr aligned_layer_B(
        new Layer<VoxelType>(layer_B.voxel_size(), layer_B.voxels_per_side()));

    Layer<VoxelType>* error_layer = nullptr;
    if (aligned_layers_and_error_layers != nullptr) {
      if (aligned_layers_and_error_layers->size() != transforms_A_B.size()) {
        aligned_layers_and_error_layers->clear();
        aligned_layers_and_error_layers->resize(transforms_A_B.size());
      }

      // Initialize and get ptr to error layer to fill out later.
      (*aligned_layers_and_error_layers)[i].second =
          typename voxblox::Layer<VoxelType>::Ptr(new voxblox::Layer<VoxelType>(
              layer_A.voxel_size(), layer_A.voxels_per_side()));
      error_layer = (*aligned_layers_and_error_layers)[i].second.get();
```

(continues on next page)

(continued from previous page)

```
      // Store the aligned object as well.
      (*aligned_layers_and_error_layers)[i].first = aligned_layer_B;
    }

    // Transform merged object into the world frame.
    transformLayer<VoxelType>(layer_B, transform_A_B, aligned_layer_B.get());

    utils::VoxelEvaluationDetails voxel_evaluation_details;
    // Evaluate the RMSE of the merged object layer in the world layer.
    utils::evaluateLayersRmse(layer_A, *aligned_layer_B, voxel_evaluation_mode,
                              &voxel_evaluation_details, error_layer);
    voxel_evaluation_details_vector->push_back(voxel_evaluation_details);
  }
}

template <typename VoxelType>
void evaluateLayerRmseAtPoses(
    const utils::VoxelEvaluationMode& voxel_evaluation_mode,
    const Layer<VoxelType>& layer_A, const Layer<VoxelType>& layer_B,
    const std::vector<Eigen::Matrix<float, 4, 4>,
                      Eigen::aligned_allocator<Eigen::Matrix<float, 4, 4>>>&
        transforms_A_B,
    std::vector<utils::VoxelEvaluationDetails>* voxel_evaluation_details_vector,
    std::vector<std::pair<typename voxblox::Layer<VoxelType>::Ptr,
                          typename voxblox::Layer<VoxelType>::Ptr>>*
        aligned_layers_and_error_layers = nullptr) {
  CHECK_NOTNULL(voxel_evaluation_details_vector);
  std::vector<Transformation> kindr_transforms_A_B;
  for (const Eigen::Matrix<float, 4, 4>& transform_A_B : transforms_A_B) {
    kindr_transforms_A_B.emplace_back(transform_A_B);
  }
  evaluateLayerRmseAtPoses(
      voxel_evaluation_mode, layer_A, layer_B, kindr_transforms_A_B,
      voxel_evaluation_details_vector, aligned_layers_and_error_layers);
}

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_MERGE_INTEGRATION_H_
```

Includes

- algorithm
- glog/logging.h
- utility
- vector
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/interpolator/interpolator.h (*File interpolator.h*)

Namespaces

- *Namespace voxblox*

File mesh.h

Contents

- *Definition (voxblox/include/voxblox/mesh/mesh.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/mesh/mesh.h)

Program Listing for File mesh.h

Return to documentation for file (voxblox/include/voxblox/mesh/mesh.h)

```
// The MIT License (MIT)
// Copyright (c) 2014 Matthew Klingensmith and Ivan Dryanovski
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef VOXBLOX_MESH_MESH_H_
#define VOXBLOX_MESH_MESH_H_

#include <cstdint>
#include <memory>

#include "voxblox/core/common.h"
```

(continues on next page)

(continued from previous page)

```
namespace voxblox {

struct Mesh {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<Mesh> Ptr;
  typedef std::shared_ptr<const Mesh> ConstPtr;

  static constexpr FloatingPoint kInvalidBlockSize = -1.0;

  Mesh()
      : block_size(kInvalidBlockSize), origin(Point::Zero()), updated(false) {
    // Do nothing.
  }

  Mesh(FloatingPoint _block_size, const Point& _origin)
      : block_size(_block_size), origin(_origin), updated(false) {
    CHECK_GT(block_size, 0.0);
  }
  virtual ~Mesh() {}

  inline bool hasVertices() const { return !vertices.empty(); }
  inline bool hasNormals() const { return !normals.empty(); }
  inline bool hasColors() const { return !colors.empty(); }
  inline bool hasTriangles() const { return !indices.empty(); }

  inline size_t size() const { return vertices.size(); }

  inline void clear() {
    vertices.clear();
    normals.clear();
    colors.clear();
    indices.clear();
  }

  inline void clearTriangles() { indices.clear(); }
  inline void clearNormals() { normals.clear(); }
  inline void clearColors() { colors.clear(); }

  inline void resize(const size_t size, const bool has_normals = true,
                     const bool has_colors = true,
                     const bool has_indices = true) {
    vertices.resize(size);

    if (has_normals) {
      normals.resize(size);
    }

    if (has_colors) {
      colors.resize(size);
    }

    if (has_indices) {
      indices.resize(size);
    }
  }
```

(continues on next page)

(continued from previous page)

```
inline void reserve(const size_t size, const bool has_normals = true,
                    const bool has_colors = true,
                    const bool has_triangles = true) {
  vertices.reserve(size);

  if (has_normals) {
    normals.reserve(size);
  }

  if (has_colors) {
    colors.reserve(size);
  }

  if (has_triangles) {
    indices.reserve(size);
  }
}

void colorizeMesh(const Color& new_color) {
  colors.clear();
  colors.resize(vertices.size(), new_color);
}

void concatenateMesh(const Mesh& other_mesh) {
  CHECK_EQ(other_mesh.hasColors(), hasColors());
  CHECK_EQ(other_mesh.hasNormals(), hasNormals());
  CHECK_EQ(other_mesh.hasTriangles(), hasTriangles());

  reserve(size() + other_mesh.size(), hasNormals(), hasColors(),
          hasTriangles());

  const size_t num_vertices_before = vertices.size();

  for (const Point& vertex : other_mesh.vertices) {
    vertices.push_back(vertex);
  }
  for (const Color& color : other_mesh.colors) {
    colors.push_back(color);
  }
  for (const Point& normal : other_mesh.normals) {
    normals.push_back(normal);
  }
  for (const size_t index : other_mesh.indices) {
    indices.push_back(index + num_vertices_before);
  }
}

Pointcloud vertices;
VertexIndexList indices;
Pointcloud normals;
Colors colors;

FloatingPoint block_size;
Point origin;

bool updated;
```

(continues on next page)

(continued from previous page)

```
};

}  // namespace voxblox

#endif  // VOXBLOX_MESH_MESH_H_
```

Includes

- cstdint
- memory
- voxblox/core/common.h (*File common.h*)

Included By

- *File sdf_ply.h*
- *File marching_cubes.h*
- *File mesh_layer.h*
- *File mesh_utils.h*
- *File conversions.h*
- *File mesh_vis.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct Mesh*

File mesh_integrator.h

Contents

- *Definition (voxblox/include/voxblox/mesh/mesh_integrator.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/mesh/mesh_integrator.h)

Program Listing for File mesh_integrator.h

Return to documentation for file (voxblox/include/voxblox/mesh/mesh_integrator.h)

```
// The MIT License (MIT)
// Copyright (c) 2014 Matthew Klingensmith and Ivan Dryanovski
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef VOXBLOX_MESH_MESH_INTEGRATOR_H_
#define VOXBLOX_MESH_MESH_INTEGRATOR_H_

#include <algorithm>
#include <list>
#include <thread>
#include <vector>

#include <glog/logging.h>
#include <Eigen/Core>

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/interpolator/interpolator.h"
#include "voxblox/mesh/marching_cubes.h"
#include "voxblox/mesh/mesh_layer.h"
#include "voxblox/utils/meshing_utils.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

struct MeshIntegratorConfig {
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  bool use_color = true;
  float min_weight = 1e-4;

  size_t integrator_threads = std::thread::hardware_concurrency();
```

(continues on next page)

(continued from previous page)

```
};

template <typename VoxelType>
class MeshIntegrator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  void initFromSdfLayer(const Layer<VoxelType>& sdf_layer) {
    voxel_size_ = sdf_layer.voxel_size();
    block_size_ = sdf_layer.block_size();
    voxels_per_side_ = sdf_layer.voxels_per_side();

    voxel_size_inv_ = 1.0 / voxel_size_;
    block_size_inv_ = 1.0 / block_size_;
    voxels_per_side_inv_ = 1.0 / voxels_per_side_;
  }

  MeshIntegrator(const MeshIntegratorConfig& config,
                 Layer<VoxelType>* sdf_layer, MeshLayer* mesh_layer)
      : config_(config),
        sdf_layer_mutable_(CHECK_NOTNULL(sdf_layer)),
        sdf_layer_const_(CHECK_NOTNULL(sdf_layer)),
        mesh_layer_(CHECK_NOTNULL(mesh_layer)) {
    initFromSdfLayer(*sdf_layer);

    cube_index_offsets_ << 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
        0, 0, 1, 1, 1, 1;

    if (config_.integrator_threads == 0) {
      LOG(WARNING) << "Automatic core count failed, defaulting to 1 threads";
      config_.integrator_threads = 1;
    }
  }

  MeshIntegrator(const MeshIntegratorConfig& config,
                 const Layer<VoxelType>& sdf_layer, MeshLayer* mesh_layer)
      : config_(config),
        sdf_layer_mutable_(nullptr),
        sdf_layer_const_(&sdf_layer),
        mesh_layer_(CHECK_NOTNULL(mesh_layer)) {
    initFromSdfLayer(sdf_layer);

    // clang-format off
    cube_index_offsets_ << 0, 1, 1, 0, 0, 1, 1, 0,
                           0, 0, 1, 1, 0, 0, 1, 1,
                           0, 0, 0, 0, 1, 1, 1, 1;
    // clang-format on

    if (config_.integrator_threads == 0) {
      LOG(WARNING) << "Automatic core count failed, defaulting to 1 threads";
      config_.integrator_threads = 1;
    }
  }

  void generateMesh(bool only_mesh_updated_blocks, bool clear_updated_flag) {
    CHECK(!clear_updated_flag || (sdf_layer_mutable_ != nullptr))
        << "If you would like to modify the updated flag in the blocks, please "
```

(continues on next page)

(continued from previous page)

```
      << "use the constructor that provides a non-const link to the sdf "
      << "layer!";
  BlockIndexList all_tsdf_blocks;
  if (only_mesh_updated_blocks) {
    sdf_layer_const_->getAllUpdatedBlocks(&all_tsdf_blocks);
  } else {
    sdf_layer_const_->getAllAllocatedBlocks(&all_tsdf_blocks);
  }

  // Allocate all the mesh memory
  for (const BlockIndex& block_index : all_tsdf_blocks) {
    mesh_layer_->allocateMeshPtrByIndex(block_index);
  }

  ThreadSafeIndex index_getter(all_tsdf_blocks.size());

  std::list<std::thread> integration_threads;
  for (size_t i = 0; i < config_.integrator_threads; ++i) {
    integration_threads.emplace_back(
        &MeshIntegrator::generateMeshBlocksFunction, this, all_tsdf_blocks,
        clear_updated_flag, &index_getter);
  }

  for (std::thread& thread : integration_threads) {
    thread.join();
  }
}

void generateMeshBlocksFunction(const BlockIndexList& all_tsdf_blocks,
                                bool clear_updated_flag,
                                ThreadSafeIndex* index_getter) {
  DCHECK(index_getter != nullptr);
  CHECK(!clear_updated_flag || (sdf_layer_mutable_ != nullptr))
      << "If you would like to modify the updated flag in the blocks, please "
      << "use the constructor that provides a non-const link to the sdf "
      << "layer!";

  size_t list_idx;
  while (index_getter->getNextIndex(&list_idx)) {
    const BlockIndex& block_idx = all_tsdf_blocks[list_idx];
    updateMeshForBlock(block_idx);
    if (clear_updated_flag) {
      typename Block<VoxelType>::Ptr block =
          sdf_layer_mutable_->getBlockPtrByIndex(block_idx);
      block->updated() = false;
    }
  }
}

void extractBlockMesh(typename Block<VoxelType>::ConstPtr block,
                      Mesh::Ptr mesh) {
  DCHECK(block != nullptr);
  DCHECK(mesh != nullptr);

  IndexElement vps = block->voxels_per_side();
  VertexIndex next_mesh_index = 0;
```

(continues on next page)

(continued from previous page)

```
    VoxelIndex voxel_index;
    for (voxel_index.x() = 0; voxel_index.x() < vps - 1; ++voxel_index.x()) {
      for (voxel_index.y() = 0; voxel_index.y() < vps - 1; ++voxel_index.y()) {
        for (voxel_index.z() = 0; voxel_index.z() < vps - 1;
             ++voxel_index.z()) {
          Point coords = block->computeCoordinatesFromVoxelIndex(voxel_index);
          extractMeshInsideBlock(*block, voxel_index, coords, &next_mesh_index,
                                 mesh.get());
        }
      }
    }

    // Max X plane
    // takes care of edge (x_max, y_max, z),
    // takes care of edge (x_max, y, z_max).
    voxel_index.x() = vps - 1;
    for (voxel_index.z() = 0; voxel_index.z() < vps; voxel_index.z()++) {
      for (voxel_index.y() = 0; voxel_index.y() < vps; voxel_index.y()++) {
        Point coords = block->computeCoordinatesFromVoxelIndex(voxel_index);
        extractMeshOnBorder(*block, voxel_index, coords, &next_mesh_index,
                            mesh.get());
      }
    }

    // Max Y plane.
    // takes care of edge (x, y_max, z_max),
    // without corner (x_max, y_max, z_max).
    voxel_index.y() = vps - 1;
    for (voxel_index.z() = 0; voxel_index.z() < vps; voxel_index.z()++) {
      for (voxel_index.x() = 0; voxel_index.x() < vps - 1; voxel_index.x()++) {
        Point coords = block->computeCoordinatesFromVoxelIndex(voxel_index);
        extractMeshOnBorder(*block, voxel_index, coords, &next_mesh_index,
                            mesh.get());
      }
    }

    // Max Z plane.
    voxel_index.z() = vps - 1;
    for (voxel_index.y() = 0; voxel_index.y() < vps - 1; voxel_index.y()++) {
      for (voxel_index.x() = 0; voxel_index.x() < vps - 1; voxel_index.x()++) {
        Point coords = block->computeCoordinatesFromVoxelIndex(voxel_index);
        extractMeshOnBorder(*block, voxel_index, coords, &next_mesh_index,
                            mesh.get());
      }
    }
  }

  virtual void updateMeshForBlock(const BlockIndex& block_index) {
    Mesh::Ptr mesh = mesh_layer_->getMeshPtrByIndex(block_index);
    mesh->clear();
    // This block should already exist, otherwise it makes no sense to update
    // the mesh for it. ;)
    typename Block<VoxelType>::ConstPtr block =
        sdf_layer_const_->getBlockPtrByIndex(block_index);

    if (!block) {
      LOG(ERROR) << "Trying to mesh a non-existent block at index: "
```

(continues on next page)

(continued from previous page)

```
                << block_index.transpose();
    return;
  }
  extractBlockMesh(block, mesh);
  // Update colors if needed.
  if (config_.use_color) {
    updateMeshColor(*block, mesh.get());
  }

  mesh->updated = true;
}

void extractMeshInsideBlock(const Block<VoxelType>& block,
                            const VoxelIndex& index, const Point& coords,
                            VertexIndex* next_mesh_index, Mesh* mesh) {
  DCHECK(next_mesh_index != nullptr);
  DCHECK(mesh != nullptr);

  Eigen::Matrix<FloatingPoint, 3, 8> cube_coord_offsets =
      cube_index_offsets_.cast<FloatingPoint>() * voxel_size_;
  Eigen::Matrix<FloatingPoint, 3, 8> corner_coords;
  Eigen::Matrix<FloatingPoint, 8, 1> corner_sdf;
  bool all_neighbors_observed = true;

  for (unsigned int i = 0; i < 8; ++i) {
    VoxelIndex corner_index = index + cube_index_offsets_.col(i);
    const VoxelType& voxel = block.getVoxelByVoxelIndex(corner_index);

    if (!utils::getSdfIfValid(voxel, config_.min_weight, &(corner_sdf(i)))) {
      all_neighbors_observed = false;
      break;
    }

    corner_coords.col(i) = coords + cube_coord_offsets.col(i);
  }

  if (all_neighbors_observed) {
    MarchingCubes::meshCube(corner_coords, corner_sdf, next_mesh_index, mesh);
  }
}

void extractMeshOnBorder(const Block<VoxelType>& block,
                         const VoxelIndex& index, const Point& coords,
                         VertexIndex* next_mesh_index, Mesh* mesh) {
  DCHECK(mesh != nullptr);

  Eigen::Matrix<FloatingPoint, 3, 8> cube_coord_offsets =
      cube_index_offsets_.cast<FloatingPoint>() * voxel_size_;
  Eigen::Matrix<FloatingPoint, 3, 8> corner_coords;
  Eigen::Matrix<FloatingPoint, 8, 1> corner_sdf;
  bool all_neighbors_observed = true;
  corner_coords.setZero();
  corner_sdf.setZero();

  for (unsigned int i = 0; i < 8; ++i) {
    VoxelIndex corner_index = index + cube_index_offsets_.col(i);
```

(continues on next page)

(continued from previous page)

```
      if (block.isValidVoxelIndex(corner_index)) {
        const VoxelType& voxel = block.getVoxelByVoxelIndex(corner_index);

        if (!utils::getSdfIfValid(voxel, config_.min_weight,
                                  &(corner_sdf(i)))) {
          all_neighbors_observed = false;
          break;
        }

        corner_coords.col(i) = coords + cube_coord_offsets.col(i);
      } else {
        // We have to access a different block.
        BlockIndex block_offset = BlockIndex::Zero();

        for (unsigned int j = 0u; j < 3u; j++) {
          if (corner_index(j) < 0) {
            block_offset(j) = -1;
            corner_index(j) = corner_index(j) + voxels_per_side_;
          } else if (corner_index(j) >=
                     static_cast<IndexElement>(voxels_per_side_)) {
            block_offset(j) = 1;
            corner_index(j) = corner_index(j) - voxels_per_side_;
          }
        }

        BlockIndex neighbor_index = block.block_index() + block_offset;

        if (sdf_layer_const_->hasBlock(neighbor_index)) {
          const Block<VoxelType>& neighbor_block =
              sdf_layer_const_->getBlockByIndex(neighbor_index);

          CHECK(neighbor_block.isValidVoxelIndex(corner_index));
          const VoxelType& voxel =
              neighbor_block.getVoxelByVoxelIndex(corner_index);

          if (!utils::getSdfIfValid(voxel, config_.min_weight,
                                    &(corner_sdf(i)))) {
            all_neighbors_observed = false;
            break;
          }

          corner_coords.col(i) = coords + cube_coord_offsets.col(i);
        } else {
          all_neighbors_observed = false;
          break;
        }
      }
    }

    if (all_neighbors_observed) {
      MarchingCubes::meshCube(corner_coords, corner_sdf, next_mesh_index, mesh);
    }
  }

  void updateMeshColor(const Block<VoxelType>& block, Mesh* mesh) {
    DCHECK(mesh != nullptr);
```

(continues on next page)

(continued from previous page)

```
    mesh->colors.clear();
    mesh->colors.resize(mesh->indices.size());

    // Use nearest-neighbor search.
    for (size_t i = 0; i < mesh->vertices.size(); i++) {
      const Point& vertex = mesh->vertices[i];
      VoxelIndex voxel_index = block.computeVoxelIndexFromCoordinates(vertex);
      if (block.isValidVoxelIndex(voxel_index)) {
        const VoxelType& voxel = block.getVoxelByVoxelIndex(voxel_index);
        utils::getColorIfValid(voxel, config_.min_weight, &(mesh->colors[i]));
      } else {
        const typename Block<VoxelType>::ConstPtr neighbor_block =
            sdf_layer_const_->getBlockPtrByCoordinates(vertex);
        const VoxelType& voxel = neighbor_block->getVoxelByCoordinates(vertex);
        utils::getColorIfValid(voxel, config_.min_weight, &(mesh->colors[i]));
      }
    }
  }

 protected:
  MeshIntegratorConfig config_;

  Layer<VoxelType>* sdf_layer_mutable_;
  const Layer<VoxelType>* sdf_layer_const_;

  MeshLayer* mesh_layer_;

  // Cached map config.
  FloatingPoint voxel_size_;
  size_t voxels_per_side_;
  FloatingPoint block_size_;

  // Derived types.
  FloatingPoint voxel_size_inv_;
  FloatingPoint voxels_per_side_inv_;
  FloatingPoint block_size_inv_;

  // Cached index map.
  Eigen::Matrix<int, 3, 8> cube_index_offsets_;
};

}  // namespace voxblox

#endif  // VOXBLOX_MESH_MESH_INTEGRATOR_H_
```

Includes

- Eigen/Core
- algorithm
- glog/logging.h
- list
- thread

- vector
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/integrator_utils.h (*File integrator_utils.h*)
- voxblox/interpolator/interpolator.h (*File interpolator.h*)
- voxblox/mesh/marching_cubes.h (*File marching_cubes.h*)
- voxblox/mesh/mesh_layer.h (*File mesh_layer.h*)
- voxblox/utils/meshing_utils.h (*File meshing_utils.h*)
- voxblox/utils/timing.h (*File timing.h*)

Included By

- *File sdf_ply.h*
- *File simulation_server.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct MeshIntegratorConfig*
- *Template Class MeshIntegrator*

File mesh_layer.h

Contents

- *Definition (voxblox/include/voxblox/mesh/mesh_layer.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/mesh/mesh_layer.h)

Program Listing for File mesh_layer.h

Return to documentation for file (voxblox/include/voxblox/mesh/mesh_layer.h)

```
#ifndef VOXBLOX_MESH_MESH_LAYER_H_
#define VOXBLOX_MESH_MESH_LAYER_H_

#include <cmath>
#include <iostream>
#include <memory>
#include <utility>
#include <vector>

#include <glog/logging.h>

#include "voxblox/core/block_hash.h"
#include "voxblox/core/common.h"
#include "voxblox/mesh/mesh.h"
#include "voxblox/mesh/mesh_utils.h"

namespace voxblox {

class MeshLayer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<MeshLayer> Ptr;
  typedef std::shared_ptr<const MeshLayer> ConstPtr;
  typedef typename AnyIndexHashMapType<Mesh::Ptr>::type MeshMap;

  explicit MeshLayer(FloatingPoint block_size)
      : block_size_(block_size), block_size_inv_(1.0 / block_size) {}
  virtual ~MeshLayer() {}

  // By index.
  inline const Mesh& getMeshByIndex(const BlockIndex& index) const {
    typename MeshMap::const_iterator it = mesh_map_.find(index);
    if (it != mesh_map_.end()) {
      return *(it->second);
    } else {
      LOG(FATAL) << "Accessed unallocated mesh at " << index.transpose();
    }
  }

  inline Mesh& getMeshByIndex(const BlockIndex& index) {
    typename MeshMap::iterator it = mesh_map_.find(index);
    if (it != mesh_map_.end()) {
      return *(it->second);
    } else {
      LOG(FATAL) << "Accessed unallocated mesh at " << index.transpose();
    }
  }

  inline typename Mesh::ConstPtr getMeshPtrByIndex(
      const BlockIndex& index) const {
    typename MeshMap::const_iterator it = mesh_map_.find(index);
    if (it != mesh_map_.end()) {
      return it->second;
    } else {
      LOG(WARNING) << "Returning null ptr to mesh!";
      return typename Mesh::ConstPtr();
```

(continues on next page)

(continued from previous page)

```
    }
  }

  inline typename Mesh::Ptr getMeshPtrByIndex(const BlockIndex& index) {
    typename MeshMap::iterator it = mesh_map_.find(index);
    if (it != mesh_map_.end()) {
      return it->second;
    } else {
      LOG(WARNING) << "Returning null ptr to mesh!";
      return typename Mesh::Ptr();
    }
  }

  inline typename Mesh::Ptr allocateMeshPtrByIndex(const BlockIndex& index) {
    typename MeshMap::iterator it = mesh_map_.find(index);
    if (it != mesh_map_.end()) {
      return it->second;
    } else {
      return allocateNewBlock(index);
    }
  }

  inline typename Mesh::ConstPtr getMeshPtrByCoordinates(
      const Point& coords) const {
    return getMeshPtrByIndex(computeBlockIndexFromCoordinates(coords));
  }

  inline typename Mesh::Ptr getMeshPtrByCoordinates(const Point& coords) {
    return getMeshPtrByIndex(computeBlockIndexFromCoordinates(coords));
  }

  inline typename Mesh::Ptr allocateMeshPtrByCoordinates(const Point& coords) {
    return allocateMeshPtrByIndex(computeBlockIndexFromCoordinates(coords));
  }

  inline BlockIndex computeBlockIndexFromCoordinates(
      const Point& coords) const {
    return getGridIndexFromPoint<BlockIndex>(coords, block_size_inv_);
  }

  typename Mesh::Ptr allocateNewBlock(const BlockIndex& index) {
    auto insert_status = mesh_map_.insert(std::make_pair(
        index, std::shared_ptr<Mesh>(new Mesh(
                   block_size_, index.cast<FloatingPoint>() * block_size_))));
    DCHECK(insert_status.second)
        << "Mesh already exists when allocating at " << index.transpose();
    DCHECK(insert_status.first->second);
    DCHECK_EQ(insert_status.first->first, index);
    return insert_status.first->second;
  }

  inline typename Mesh::Ptr allocateNewBlockByCoordinates(const Point& coords) {
    return allocateNewBlock(computeBlockIndexFromCoordinates(coords));
  }

  void removeMesh(const BlockIndex& index) { mesh_map_.erase(index); }
```

(continues on next page)

(continued from previous page)

```
void removeMeshByCoordinates(const Point& coords) {
  mesh_map_.erase(computeBlockIndexFromCoordinates(coords));
}

void clearDistantMesh(const Point& center, const double max_distance) {
  // we clear the mesh, but do not delete it from the map as the empty mesh
  // must be sent to rviz so it is also cleared there
  for (std::pair<const BlockIndex, typename Mesh::Ptr>& kv : mesh_map_) {
    if ((kv.second->origin - center).squaredNorm() >
        max_distance * max_distance) {
      kv.second->clear();
      kv.second->updated = true;
    }
  }
}

void getAllAllocatedMeshes(BlockIndexList* meshes) const {
  meshes->clear();
  meshes->reserve(mesh_map_.size());
  for (const std::pair<const BlockIndex, typename Mesh::Ptr>& kv :
       mesh_map_) {
    meshes->emplace_back(kv.first);
  }
}

void getAllUpdatedMeshes(BlockIndexList* meshes) const {
  meshes->clear();
  for (const std::pair<const BlockIndex, typename Mesh::Ptr>& kv :
       mesh_map_) {
    if (kv.second->updated) {
      meshes->emplace_back(kv.first);
    }
  }
}

void getMesh(Mesh* combined_mesh) const {
  CHECK_NOTNULL(combined_mesh);

  // Combine everything in the layer into one giant combined mesh.

  BlockIndexList mesh_indices;
  getAllAllocatedMeshes(&mesh_indices);

  // Check if color, normals and indices are enabled for the first non-empty
  // mesh. If they are, they need to be enabled for all other ones as well.
  bool has_colors = false;
  bool has_normals = false;
  bool has_indices = false;
  if (!mesh_indices.empty()) {
    for (const BlockIndex& block_index : mesh_indices) {
      Mesh::ConstPtr mesh = getMeshPtrByIndex(block_index);
      if (!mesh->vertices.empty()) {
        has_colors = mesh->hasColors();
        has_normals = mesh->hasNormals();
        has_indices = mesh->hasTriangles();
        break;
      }
```

(continues on next page)

(continued from previous page)

```
    }
  }

  // Loop again over all meshes to figure out how big the mesh needs to be.
  size_t mesh_size = 0;
  for (const BlockIndex& block_index : mesh_indices) {
    Mesh::ConstPtr mesh = getMeshPtrByIndex(block_index);
    mesh_size += mesh->vertices.size();
  }

  // Reserve space for the mesh.
  combined_mesh->reserve(mesh_size, has_normals, has_colors, has_indices);

  size_t new_index = 0u;
  for (const BlockIndex& block_index : mesh_indices) {
    Mesh::ConstPtr mesh = getMeshPtrByIndex(block_index);

    // Check assumption that all meshes have same configuration regarding
    // colors, normals and indices.
    if (!mesh->vertices.empty()) {
      CHECK_EQ(has_colors, mesh->hasColors());
      CHECK_EQ(has_normals, mesh->hasNormals());
      CHECK_EQ(has_indices, mesh->hasTriangles());
    }

    // Copy the mesh content into the combined mesh. This is done in triplets
    // for readability only, as one loop iteration will then copy one
    // triangle.
    for (size_t i = 0; i < mesh->vertices.size(); i += 3, new_index += 3) {
      CHECK_LT(new_index + 2, mesh_size);

      combined_mesh->vertices.push_back(mesh->vertices[i]);
      combined_mesh->vertices.push_back(mesh->vertices[i + 1]);
      combined_mesh->vertices.push_back(mesh->vertices[i + 2]);

      if (has_colors) {
        combined_mesh->colors.push_back(mesh->colors[i]);
        combined_mesh->colors.push_back(mesh->colors[i + 1]);
        combined_mesh->colors.push_back(mesh->colors[i + 2]);
      }
      if (has_normals) {
        combined_mesh->normals.push_back(mesh->normals[i]);
        combined_mesh->normals.push_back(mesh->normals[i + 1]);
        combined_mesh->normals.push_back(mesh->normals[i + 2]);
      }
      if (has_indices) {
        combined_mesh->indices.push_back(new_index);
        combined_mesh->indices.push_back(new_index + 1);
        combined_mesh->indices.push_back(new_index + 2);
      }
    }
  }

  // Verify combined mesh.
  if (combined_mesh->hasColors()) {
    CHECK_EQ(combined_mesh->vertices.size(), combined_mesh->colors.size());
  }
```

(continues on next page)

(continued from previous page)

```
    if (combined_mesh->hasNormals()) {
      CHECK_EQ(combined_mesh->vertices.size(), combined_mesh->normals.size());
    }

    CHECK_EQ(combined_mesh->vertices.size(), combined_mesh->indices.size());
  }

  void getConnectedMesh(
      Mesh* connected_mesh,
      const FloatingPoint approximate_vertex_proximity_threshold =
          1e-10) const {
    BlockIndexList mesh_indices;
    getAllAllocatedMeshes(&mesh_indices);

    AlignedVector<Mesh::ConstPtr> meshes;
    meshes.reserve(mesh_indices.size());
    for (const BlockIndex& block_index : mesh_indices) {
      meshes.push_back(getMeshPtrByIndex(block_index));
    }

    createConnectedMesh(meshes, connected_mesh,
                        approximate_vertex_proximity_threshold);
  }

  size_t getNumberOfAllocatedMeshes() const { return mesh_map_.size(); }

  void clear() { mesh_map_.clear(); }

  FloatingPoint block_size() const { return block_size_; }

  FloatingPoint block_size_inv() const { return block_size_inv_; }

 private:
  FloatingPoint block_size_;

  // Derived types.
  FloatingPoint block_size_inv_;

  MeshMap mesh_map_;
};

}  // namespace voxblox

#endif  // VOXBLOX_MESH_MESH_LAYER_H_
```

Includes

- cmath
- glog/logging.h
- iostream
- memory
- utility

- vector
- voxblox/core/block_hash.h (*File block_hash.h*)
- voxblox/core/common.h (*File common.h*)
- voxblox/mesh/mesh.h (*File mesh.h*)
- voxblox/mesh/mesh_utils.h (*File mesh_utils.h*)

Included By

- *File mesh_ply.h*
- *File sdf_ply.h*
- *File mesh_integrator.h*
- *File mesh_pcl.h*
- *File mesh_vis.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class MeshLayer*

File mesh_pcl.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/mesh_pcl.h)*
- *Includes*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/mesh_pcl.h)

Program Listing for File mesh_pcl.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/mesh_pcl.h)

```
// The MIT License (MIT)
// Copyright (c) 2014 Matthew Klingensmith and Ivan Dryanovski
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
```

(continues on next page)

(continued from previous page)

```
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
// Mesh output taken from open_chisel: github.com/personalrobotics/OpenChisel

#ifndef VOXBLOX_ROS_MESH_PCL_H_
#define VOXBLOX_ROS_MESH_PCL_H_

#include <string>
#include <vector>

#include <pcl/point_types.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl_ros/point_cloud.h>

#include <voxblox/core/common.h>
#include <voxblox/mesh/mesh_layer.h>

namespace voxblox {

inline void toPCLPolygonMesh(
    const MeshLayer& mesh_layer, const std::string frame_id,
    pcl::PolygonMesh* polygon_mesh_ptr,
    const bool simplify_and_connect_mesh = true,
    const FloatingPoint vertex_proximity_threshold = 1e-10) {
  CHECK_NOTNULL(polygon_mesh_ptr);

  // Constructing the vertices pointcloud
  pcl::PointCloud<pcl::PointXYZ> pointcloud;
  std::vector<pcl::Vertices> polygons;

  Mesh mesh;
  convertMeshLayerToMesh(mesh_layer, &mesh, simplify_and_connect_mesh,
                         vertex_proximity_threshold);

  // add points
  pointcloud.reserve(mesh.vertices.size());
  for (const Point& point : mesh.vertices) {
    pointcloud.push_back(pcl::PointXYZ(static_cast<float>(point[0]),
                                       static_cast<float>(point[1]),
                                       static_cast<float>(point[2])));
  }
  // add triangles
  pcl::Vertices vertices_idx;
  polygons.reserve(mesh.indices.size() / 3);
```

(continues on next page)

(continued from previous page)

```
  for (const VertexIndex& idx : mesh.indices) {
    vertices_idx.vertices.push_back(idx);

    if (vertices_idx.vertices.size() == 3) {
      polygons.push_back(vertices_idx);
      vertices_idx.vertices.clear();
    }
  }

  // Converting to the pointcloud binary
  pcl::PCLPointCloud2 pointcloud2;
  pcl::toPCLPointCloud2(pointcloud, pointcloud2);
  // Filling the mesh
  polygon_mesh_ptr->header.frame_id = frame_id;
  polygon_mesh_ptr->cloud = pointcloud2;
  polygon_mesh_ptr->polygons = polygons;
}

inline void toSimplifiedPCLPolygonMesh(
    const MeshLayer& mesh_layer, const std::string frame_id,
    const FloatingPoint vertex_proximity_threshold,
    pcl::PolygonMesh* polygon_mesh_ptr) {
  constexpr bool kSimplifiedAndConnectedMesh = true;
  toPCLPolygonMesh(mesh_layer, frame_id, polygon_mesh_ptr,
                   kSimplifiedAndConnectedMesh, vertex_proximity_threshold);
}

inline void toConnectedPCLPolygonMesh(const MeshLayer& mesh_layer,
                                      const std::string frame_id,
                                      pcl::PolygonMesh* polygon_mesh_ptr) {
  constexpr bool kSimplifiedAndConnectedMesh = true;
  constexpr FloatingPoint kVertexThreshold = 1e-10;
  toPCLPolygonMesh(mesh_layer, frame_id, polygon_mesh_ptr,
                   kSimplifiedAndConnectedMesh, kVertexThreshold);
}

}  // namespace voxblox

#endif  // VOXBLOX_ROS_MESH_PCL_H_
```

Includes

- pcl/point_types.h
- pcl_conversions/pcl_conversions.h
- pcl_ros/point_cloud.h
- string
- vector
- voxblox/core/common.h (*File common.h*)
- voxblox/mesh/mesh_layer.h (*File mesh_layer.h*)

Namespaces

- *Namespace voxblox*

File mesh_ply.h

Contents

- *Definition (voxblox/include/voxblox/io/mesh_ply.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/io/mesh_ply.h)

Program Listing for File mesh_ply.h

Return to documentation for file (voxblox/include/voxblox/io/mesh_ply.h)

```
// NOTE: From open_chisel: github.com/personalrobotics/OpenChisel/
// The MIT License (MIT)
// Copyright (c) 2014 Matthew Klingensmith and Ivan Dryanovski
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.

#ifndef VOXBLOX_IO_MESH_PLY_H_
#define VOXBLOX_IO_MESH_PLY_H_

#include "voxblox/mesh/mesh_layer.h"

#include <fstream>
#include <iostream>
#include <string>
```

(continues on next page)

(continued from previous page)

```
namespace voxblox {

bool convertMeshLayerToMesh(
    const MeshLayer& mesh_layer, Mesh* mesh, const bool connected_mesh = true,
    const FloatingPoint vertex_proximity_threshold = 1e-10);

bool outputMeshLayerAsPly(const std::string& filename,
                          const MeshLayer& mesh_layer);

bool outputMeshLayerAsPly(const std::string& filename,
                          const bool connected_mesh,
                          const MeshLayer& mesh_layer);

bool outputMeshAsPly(const std::string& filename, const Mesh& mesh);

}  // namespace voxblox

#endif  // VOXBLOX_IO_MESH_PLY_H_
```

Includes

- fstream
- iostream
- string
- voxblox/mesh/mesh_layer.h (*File mesh_layer.h*)

Included By

- *File sdf_ply.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

File mesh_utils.h

Contents

- *Definition (voxblox/include/voxblox/mesh/mesh_utils.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/mesh/mesh_utils.h)

Program Listing for File mesh_utils.h

Return to documentation for file (voxblox/include/voxblox/mesh/mesh_utils.h)

```
#ifndef VOXBLOX_MESH_MESH_UTILS_H_
#define VOXBLOX_MESH_MESH_UTILS_H_

#include <vector>

#include "voxblox/core/block_hash.h"
#include "voxblox/core/common.h"
#include "voxblox/mesh/mesh.h"

namespace voxblox {

inline void createConnectedMesh(
    const AlignedVector<Mesh::ConstPtr>& meshes, Mesh* connected_mesh,
    const FloatingPoint approximate_vertex_proximity_threshold = 1e-10) {
  CHECK_NOTNULL(connected_mesh);

  // Used to prevent double ups in vertices. We need to use a long long based
  // index, to prevent overflows.
  LongIndexHashMapType<size_t>::type uniques;

  const double threshold_inv =
      1. / static_cast<double>(approximate_vertex_proximity_threshold);

  // Combine everything in the layer into one giant combined mesh.
  size_t new_vertex_index = 0u;
  for (const Mesh::ConstPtr& mesh : meshes) {
    // Skip empty meshes.
    if (mesh->vertices.empty()) {
      continue;
    }

    // Make sure there are 3 distinct vertices for every triangle before
    // merging.
    CHECK_EQ(mesh->vertices.size(), mesh->indices.size());
    CHECK_EQ(mesh->vertices.size() % 3u, 0u);
    CHECK_EQ(mesh->indices.size() % 3u, 0u);

    // Stores the mapping from old vertex index to the new one in the combined
    // mesh. This is later used to adapt the triangles to the new, global
    // indexing of the combined mesh.
    std::vector<size_t> old_to_new_indices;
    old_to_new_indices.resize(mesh->vertices.size());

    size_t new_num_vertices_from_this_block = 0u;
    for (size_t old_vertex_idx = 0u; old_vertex_idx < mesh->vertices.size();
         ++old_vertex_idx) {
      // We scale the vertices by the inverse of the merging tolerance and
      // then compute a discretized grid index in that scale.
      // This exhibits the behaviour of merging two vertices that are
      // closer than the threshold.
      CHECK_LT(old_vertex_idx, mesh->vertices.size());
```

(continues on next page)

(continued from previous page)

```
      const Point vertex = mesh->vertices[old_vertex_idx];
      const Eigen::Vector3d scaled_vector =
          vertex.cast<double>() * threshold_inv;
      const LongIndex vertex_3D_index = LongIndex(
          std::round(scaled_vector.x()), std::round(scaled_vector.y()),
          std::round(scaled_vector.z()));

      // If the current vertex falls into the same grid cell as a previous
      // vertex, we merge them. This is done by assigning the new vertex to
      // the same vertex index as the first vertex that fell into that cell.

      LongIndexHashMapType<size_t>::type::const_iterator it =
          uniques.find(vertex_3D_index);

      const bool vertex_is_unique_so_far = (it == uniques.end());
      if (vertex_is_unique_so_far) {
        // Copy vertex and associated data to combined mesh.
        connected_mesh->vertices.push_back(vertex);

        if (mesh->hasColors()) {
          CHECK_LT(old_vertex_idx, mesh->colors.size());
          connected_mesh->colors.push_back(mesh->colors[old_vertex_idx]);
        }
        if (mesh->hasNormals()) {
          CHECK_LT(old_vertex_idx, mesh->normals.size());
          connected_mesh->normals.push_back(mesh->normals[old_vertex_idx]);
        }

        // Store the new vertex index in the unique-vertex-map to be able to
        // retrieve this index later if we encounter vertices that are
        // supposed to be merged with this one.
        CHECK(uniques.emplace(vertex_3D_index, new_vertex_index).second);

        // Also store a mapping from old index to new index for this mesh
        // block to later adapt the triangle indexing.
        CHECK_LT(old_vertex_idx, old_to_new_indices.size());
        old_to_new_indices[old_vertex_idx] = new_vertex_index;

        ++new_vertex_index;
        ++new_num_vertices_from_this_block;
      } else {
        // If this vertex is not unique, we map it's vertex index to the new
        // vertex index.
        CHECK_LT(old_vertex_idx, old_to_new_indices.size());
        old_to_new_indices[old_vertex_idx] = it->second;

        // Add all normals (this will average them once they are renormalized
        // later)
        connected_mesh->normals[it->second] += mesh->normals[old_vertex_idx];
      }

      // Make sure the indexing is correct.
      CHECK_EQ(connected_mesh->vertices.size(), new_vertex_index);
    }

    // Renormalize normals
    for (Point& normal : connected_mesh->normals) {
```

(continues on next page)

(continued from previous page)

```
      FloatingPoint length = normal.norm();
      if (length > kEpsilon) {
        normal /= length;
      } else {
        normal = Point(0.0f, 0.0f, 1.0f);
      }
    }

    // Make sure we have a mapping for every old vertex index.
    CHECK_EQ(old_to_new_indices.size(), mesh->vertices.size());

    // Append triangles and adjust their indices if necessary.
    // We discard triangles where all old vertices were mapped to the same
    // vertex.
    size_t new_num_triangle_from_this_block = 0u;
    for (size_t triangle_idx = 0u; triangle_idx < mesh->indices.size();
         triangle_idx += 3u) {
      CHECK_LT(triangle_idx + 2u, mesh->indices.size());

      // Retrieve old vertex indices.
      size_t vertex_0 = mesh->indices[triangle_idx];
      size_t vertex_1 = mesh->indices[triangle_idx + 1u];
      size_t vertex_2 = mesh->indices[triangle_idx + 2u];

      // Make sure the old indices were valid before remapping.
      CHECK_LT(vertex_0, old_to_new_indices.size());
      CHECK_LT(vertex_1, old_to_new_indices.size());
      CHECK_LT(vertex_2, old_to_new_indices.size());

      // Apply vertex index mapping.
      vertex_0 = old_to_new_indices[vertex_0];
      vertex_1 = old_to_new_indices[vertex_1];
      vertex_2 = old_to_new_indices[vertex_2];

      // Make sure the new indices are valid after remapping.
      CHECK_LT(vertex_0, new_vertex_index);
      CHECK_LT(vertex_1, new_vertex_index);
      CHECK_LT(vertex_2, new_vertex_index);

      // Get rid of triangles where all two or three vertices have been
      // merged.
      const bool two_or_three_vertex_indices_are_the_same =
          (vertex_0 == vertex_1) || (vertex_1 == vertex_2) ||
          (vertex_0 == vertex_2);

      if (!two_or_three_vertex_indices_are_the_same) {
        connected_mesh->indices.push_back(vertex_0);
        connected_mesh->indices.push_back(vertex_1);
        connected_mesh->indices.push_back(vertex_2);
        ++new_num_triangle_from_this_block;
      }
    }
  }

  // Verify combined mesh.
  if (connected_mesh->hasColors()) {
    CHECK_EQ(connected_mesh->vertices.size(), connected_mesh->colors.size());
```

(continues on next page)

(continued from previous page)

```
  }
  if (connected_mesh->hasNormals()) {
    CHECK_EQ(connected_mesh->vertices.size(), connected_mesh->normals.size());
  }
}

inline void createConnectedMesh(
    const Mesh& mesh, Mesh* connected_mesh,
    const FloatingPoint approximate_vertex_proximity_threshold = 1e-10) {
  AlignedVector<Mesh::ConstPtr> meshes;
  meshes.push_back(Mesh::ConstPtr(&mesh, [](Mesh const*) {}));
  createConnectedMesh(meshes, connected_mesh,
                      approximate_vertex_proximity_threshold);
}

};  // namespace voxblox

#endif  // VOXBLOX_MESH_MESH_UTILS_H_
```

Includes

- `vector`
- `voxblox/core/block_hash.h` (*File block_hash.h*)
- `voxblox/core/common.h` (*File common.h*)
- `voxblox/mesh/mesh.h` (*File mesh.h*)

Included By

- *File mesh_layer.h*

Namespaces

- *Namespace voxblox*

File mesh_vis.h

Contents

- *Definition (`voxblox_ros/include/voxblox_ros/mesh_vis.h`)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/mesh_vis.h)

Program Listing for File mesh_vis.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/mesh_vis.h)

```
// The MIT License (MIT)
// Copyright (c) 2014 Matthew Klingensmith and Ivan Dryanovski
//
// Permission is hereby granted, free of charge, to any person obtaining a copy
// of this software and associated documentation files (the "Software"), to deal
// in the Software without restriction, including without limitation the rights
// to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
// copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in
// all
// copies or substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
// LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
// SOFTWARE.
// Mesh output taken from open_chisel: github.com/personalrobotics/OpenChisel

#ifndef VOXBLOX_ROS_MESH_VIS_H_
#define VOXBLOX_ROS_MESH_VIS_H_

#include <eigen_conversions/eigen_msg.h>
#include <visualization_msgs/Marker.h>
#include <algorithm>
#include <limits>

#include <voxblox/core/common.h>
#include <voxblox/integrator/esdf_integrator.h>
#include <voxblox/integrator/tsdf_integrator.h>
#include <voxblox/mesh/mesh.h>
#include <voxblox/mesh/mesh_layer.h>
#include <voxblox_msgs/Mesh.h>

#include "voxblox_ros/conversions.h"

namespace voxblox {

enum ColorMode {
  kColor = 0,
  kHeight,
  kNormals,
  kGray,
  kLambert,
  kLambertColor
};
```

(continues on next page)

(continued from previous page)

```
inline Point lambertShading(const Point& normal, const Point& light,
                            const Point& color) {
  return std::max<FloatingPoint>(normal.dot(light), 0.0f) * color;
}

inline void lambertColorFromColorAndNormal(const Color& color,
                                           const Point& normal,
                                           std_msgs::ColorRGBA* color_msg) {
  // These are just some arbitrary light directions, I believe taken from
  // OpenChisel.
  const Point light_dir = Point(0.8f, -0.2f, 0.7f).normalized();
  const Point light_dir2 = Point(-0.5f, 0.2f, 0.2f).normalized();
  const Point ambient(0.2f, 0.2f, 0.2f);
  const Point color_pt(color.r / 255.0, color.g / 255.0, color.b / 255.0);

  Point lambert = lambertShading(normal, light_dir, color_pt) +
                  lambertShading(normal, light_dir2, color_pt) + ambient;

  color_msg->r = std::min<FloatingPoint>(lambert.x(), 1.0);
  color_msg->g = std::min<FloatingPoint>(lambert.y(), 1.0);
  color_msg->b = std::min<FloatingPoint>(lambert.z(), 1.0);
  color_msg->a = 1.0;
}

inline void lambertColorFromNormal(const Point& normal,
                                   std_msgs::ColorRGBA* color_msg) {
  lambertColorFromColorAndNormal(Color(127, 127, 127), normal, color_msg);
}

inline void normalColorFromNormal(const Point& normal,
                                  std_msgs::ColorRGBA* color_msg) {
  // Normals should be in the scale -1 to 1, so we need to shift them to
  // 0 -> 1 range.
  color_msg->r = normal.x() * 0.5 + 0.5;
  color_msg->g = normal.y() * 0.5 + 0.5;
  color_msg->b = normal.z() * 0.5 + 0.5;
  color_msg->a = 1.0;
}

inline void heightColorFromVertex(const Point& vertex,
                                  std_msgs::ColorRGBA* color_msg) {
  // TODO(helenol): figure out a nicer way to do this without hard-coded
  // constants.
  const double min_z = -1;
  const double max_z = 10;
  double mapped_height = std::min<FloatingPoint>(
      std::max<FloatingPoint>((vertex.z() - min_z) / (max_z - min_z), 0.0),
      1.0);
  colorVoxbloxToMsg(rainbowColorMap(mapped_height), color_msg);
}

inline std_msgs::ColorRGBA getVertexColor(const Mesh::ConstPtr& mesh,
                                          const ColorMode& color_mode,
                                          const size_t index) {
  std_msgs::ColorRGBA color_msg;
  switch (color_mode) {
    case kColor:
```

(continues on next page)

(continued from previous page)

```
      colorVoxbloxToMsg(mesh->colors[index], &color_msg);
      break;
    case kHeight:
      heightColorFromVertex(mesh->vertices[index], &color_msg);
      break;
    case kNormals:
      normalColorFromNormal(mesh->normals[index], &color_msg);
      break;
    case kLambert:
      lambertColorFromNormal(mesh->normals[index], &color_msg);
      break;
    case kLambertColor:
      lambertColorFromColorAndNormal(mesh->colors[index], mesh->normals[index],
                                     &color_msg);
      break;
    case kGray:
      color_msg.r = color_msg.g = color_msg.b = 0.5;
      color_msg.a = 1.0;
      break;
  }
  return color_msg;
}

inline void generateVoxbloxMeshMsg(const MeshLayer::Ptr& mesh_layer,
                                   ColorMode color_mode,
                                   voxblox_msgs::Mesh* mesh_msg) {
  CHECK_NOTNULL(mesh_msg);
  mesh_msg->header.stamp = ros::Time::now();

  BlockIndexList mesh_indices;
  mesh_layer->getAllUpdatedMeshes(&mesh_indices);

  mesh_msg->block_edge_length = mesh_layer->block_size();
  mesh_msg->mesh_blocks.reserve(mesh_indices.size());

  for (const BlockIndex& block_index : mesh_indices) {
    Mesh::Ptr mesh = mesh_layer->getMeshPtrByIndex(block_index);

    voxblox_msgs::MeshBlock mesh_block;
    mesh_block.index[0] = block_index.x();
    mesh_block.index[1] = block_index.y();
    mesh_block.index[2] = block_index.z();

    mesh_block.x.reserve(mesh->vertices.size());
    mesh_block.y.reserve(mesh->vertices.size());
    mesh_block.z.reserve(mesh->vertices.size());

    // normal coloring is used by RViz plugin by default, so no need to send it
    if (color_mode != kNormals) {
      mesh_block.r.reserve(mesh->vertices.size());
      mesh_block.g.reserve(mesh->vertices.size());
      mesh_block.b.reserve(mesh->vertices.size());
    }
    for (size_t i = 0u; i < mesh->vertices.size(); ++i) {
      // We convert from an absolute global frame to a normalized local frame.
      // Each vertex is given as its distance from the blocks origin in units of
      // (2*block_size). This results in all points obtaining a value in the
```

(continues on next page)

(continued from previous page)

```
      // range 0 to 1. To enforce this 0 to 1 range we technically only need to
      // divide by (block_size + voxel_size). The + voxel_size comes from the
      // way marching cubes allows the mesh to interpolate between this and a
      // neighboring block. We instead divide by (block_size + block_size) as
      // the mesh layer has no knowledge of how many voxels are inside a block.
      const Point normalized_verticies =
          0.5f * (mesh_layer->block_size_inv() * mesh->vertices[i] -
                  block_index.cast<FloatingPoint>());

      // check all points are in range [0, 1.0]
      CHECK_LE(normalized_verticies.squaredNorm(), 1.0f);
      CHECK((normalized_verticies.array() >= 0.0).all());

      // convert to uint16_t fixed point representation
      mesh_block.x.push_back(std::numeric_limits<uint16_t>::max() *
                             normalized_verticies.x());
      mesh_block.y.push_back(std::numeric_limits<uint16_t>::max() *
                             normalized_verticies.y());
      mesh_block.z.push_back(std::numeric_limits<uint16_t>::max() *
                             normalized_verticies.z());

      if (color_mode != kNormals) {
        const std_msgs::ColorRGBA color_msg =
            getVertexColor(mesh, color_mode, i);
        mesh_block.r.push_back(std::numeric_limits<uint8_t>::max() *
                               color_msg.r);
        mesh_block.g.push_back(std::numeric_limits<uint8_t>::max() *
                               color_msg.g);
        mesh_block.b.push_back(std::numeric_limits<uint8_t>::max() *
                               color_msg.b);
      }
    }

    mesh_msg->mesh_blocks.push_back(mesh_block);

    // delete empty mesh blocks after sending them
    if (!mesh->hasVertices()) {
      mesh_layer->removeMesh(block_index);
    }

    mesh->updated = false;
  }
}

inline void fillMarkerWithMesh(const MeshLayer::ConstPtr& mesh_layer,
                               ColorMode color_mode,
                               visualization_msgs::Marker* marker) {
  CHECK_NOTNULL(marker);
  marker->header.stamp = ros::Time::now();
  marker->ns = "mesh";
  marker->scale.x = 1;
  marker->scale.y = 1;
  marker->scale.z = 1;
  marker->pose.orientation.x = 0;
  marker->pose.orientation.y = 0;
  marker->pose.orientation.z = 0;
  marker->pose.orientation.w = 1;
```

(continues on next page)

(continued from previous page)

```
  marker->type = visualization_msgs::Marker::TRIANGLE_LIST;

  BlockIndexList mesh_indices;
  mesh_layer->getAllAllocatedMeshes(&mesh_indices);

  for (const BlockIndex& block_index : mesh_indices) {
    Mesh::ConstPtr mesh = mesh_layer->getMeshPtrByIndex(block_index);

    if (!mesh->hasVertices()) {
      continue;
    }
    // Check that we can actually do the color stuff.
    if (color_mode == kColor || color_mode == kLambertColor) {
      CHECK(mesh->hasColors());
    }
    if (color_mode == kNormals || color_mode == kLambert ||
        color_mode == kLambertColor) {
      CHECK(mesh->hasNormals());
    }

    for (size_t i = 0u; i < mesh->vertices.size(); i++) {
      geometry_msgs::Point point_msg;
      tf::pointEigenToMsg(mesh->vertices[i].cast<double>(), point_msg);
      marker->points.push_back(point_msg);
      marker->colors.push_back(getVertexColor(mesh, color_mode, i));
    }
  }
}

inline void fillPointcloudWithMesh(
    const MeshLayer::ConstPtr& mesh_layer, ColorMode color_mode,
    pcl::PointCloud<pcl::PointXYZRGB>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  pointcloud->clear();

  BlockIndexList mesh_indices;
  mesh_layer->getAllAllocatedMeshes(&mesh_indices);

  for (const BlockIndex& block_index : mesh_indices) {
    Mesh::ConstPtr mesh = mesh_layer->getMeshPtrByIndex(block_index);

    if (!mesh->hasVertices()) {
      continue;
    }
    // Check that we can actually do the color stuff.
    if (color_mode == kColor || color_mode == kLambertColor) {
      CHECK(mesh->hasColors());
    }
    if (color_mode == kNormals || color_mode == kLambert ||
        color_mode == kLambertColor) {
      CHECK(mesh->hasNormals());
    }

    for (size_t i = 0u; i < mesh->vertices.size(); i++) {
      pcl::PointXYZRGB point;
      point.x = mesh->vertices[i].x();
      point.y = mesh->vertices[i].y();
```

(continues on next page)

(continued from previous page)

```
      point.z = mesh->vertices[i].z();

      Color color;
      colorMsgToVoxblox(getVertexColor(mesh, color_mode, i), &color);
      point.r = color.r;
      point.g = color.g;
      point.b = color.b;

      pointcloud->push_back(point);
    }
  }
}

}  // namespace voxblox

#endif  // VOXBLOX_ROS_MESH_VIS_H_
```

Includes

- algorithm
- eigen_conversions/eigen_msg.h
- limits
- visualization_msgs/Marker.h
- voxblox/core/common.h (*File common.h*)
- voxblox/integrator/esdf_integrator.h (*File esdf_integrator.h*)
- voxblox/integrator/tsdf_integrator.h (*File tsdf_integrator.h*)
- voxblox/mesh/mesh.h (*File mesh.h*)
- voxblox/mesh/mesh_layer.h (*File mesh_layer.h*)
- voxblox_msgs/Mesh.h
- voxblox_ros/conversions.h (*File conversions.h*)

Included By

- *File intensity_vis.h*
- *File simulation_server.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

File meshing_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/meshing_utils.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/meshing_utils.h)

Program Listing for File meshing_utils.h

Return to documentation for file (voxblox/include/voxblox/utils/meshing_utils.h)

```
#ifndef VOXBLOX_UTILS_MESHING_UTILS_H_
#define VOXBLOX_UTILS_MESHING_UTILS_H_

#include "voxblox/core/common.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
namespace utils {

template <typename VoxelType>
bool getSdfIfValid(const VoxelType& voxel, const FloatingPoint min_weight,
                   FloatingPoint* sdf);

template <>
inline bool getSdfIfValid(const TsdfVoxel& voxel,
                          const FloatingPoint min_weight, FloatingPoint* sdf) {
  DCHECK(sdf != nullptr);
  if (voxel.weight <= min_weight) {
    return false;
  }
  *sdf = voxel.distance;
  return true;
}

template <>
inline bool getSdfIfValid(const EsdfVoxel& voxel,
                          const FloatingPoint /*min_weight*/,
                          FloatingPoint* sdf) {
  DCHECK(sdf != nullptr);
  if (!voxel.observed) {
    return false;
  }
  *sdf = voxel.distance;
  return true;
}

template <typename VoxelType>
```

(continues on next page)

(continued from previous page)

```
bool getColorIfValid(const VoxelType& voxel, const FloatingPoint min_weight,
                     Color* color);

template <>
inline bool getColorIfValid(const TsdfVoxel& voxel,
                            const FloatingPoint min_weight, Color* color) {
  DCHECK(color != nullptr);
  if (voxel.weight <= min_weight) {
    return false;
  }
  *color = voxel.color;
  return true;
}

template <>
inline bool getColorIfValid(const EsdfVoxel& voxel,
                            const FloatingPoint /*min_weight*/, Color* color) {
  DCHECK(color != nullptr);
  if (!voxel.observed) {
    return false;
  }
  *color = Color(255u, 255u, 255u);
  return true;
}

}  // namespace utils
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_MESHING_UTILS_H_
```

Includes

- `voxblox/core/common.h` (*File common.h*)
- `voxblox/core/voxel.h` (*File voxel.h*)

Included By

- *File mesh_integrator.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::utils*

File neighbor_tools.h

Contents

- *Definition (voxblox/include/voxblox/utils/neighbor_tools.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/utils/neighbor_tools.h)

Program Listing for File neighbor_tools.h

Return to documentation for file (voxblox/include/voxblox/utils/neighbor_tools.h)

```
#ifndef VOXBLOX_UTILS_NEIGHBOR_TOOLS_H_
#define VOXBLOX_UTILS_NEIGHBOR_TOOLS_H_

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"

namespace voxblox {

enum Connectivity : unsigned int {
  kSix = 6u,
  kEighteen = 18u,
  kTwentySix = 26u
};

class NeighborhoodLookupTables {
 public:
  typedef Eigen::Matrix<LongIndexElement, 3, Connectivity::kTwentySix>
      LongIndexOffsets;
  typedef Eigen::Matrix<IndexElement, 3, Connectivity::kTwentySix> IndexOffsets;
  typedef Eigen::Matrix<float, 1, Connectivity::kTwentySix> Distances;

  /*
   * Stores the distances to the 6, 18, and 26 neighborhood, in that order.
   * These distances need to be scaled by the voxel distance to get metric
   * distances.
   */
  static const Distances kDistances;

  /*
   * Lookup table for the offsets between a index and its 6, 18, and 26
   * neighborhood, in that order. These two offset tables are the same except
   * for the type, this saves casting the offset when used with either global
   * index (long) or local index (int) in the neighborhood lookup.
   */
  static const IndexOffsets kOffsets;
  static const LongIndexOffsets kLongOffsets;
};

template <Connectivity kConnectivity = Connectivity::kTwentySix>
class Neighborhood : public NeighborhoodLookupTables {
```

(continues on next page)

(continued from previous page)

```
public:
 typedef Eigen::Matrix<LongIndexElement, 3, kConnectivity> IndexMatrix;

 static void getFromGlobalIndex(const GlobalIndex& global_index,
                                IndexMatrix* neighbors) {
   CHECK_NOTNULL(neighbors);
   for (unsigned int i = 0u; i < kConnectivity; ++i) {
     neighbors->col(i) = global_index + kLongOffsets.col(i);
   }
 }

 static void getFromBlockAndVoxelIndexAndDirection(
     const BlockIndex& block_index, const VoxelIndex& voxel_index,
     const SignedIndex& direction, const size_t voxels_per_side,
     BlockIndex* neighbor_block_index, VoxelIndex* neighbor_voxel_index) {
   CHECK_GT(voxels_per_side, 0u);
   CHECK_NOTNULL(neighbor_block_index);
   CHECK_NOTNULL(neighbor_voxel_index);

   *neighbor_block_index = block_index;
   *neighbor_voxel_index = voxel_index + direction;

   for (unsigned int i = 0u; i < 3u; ++i) {
     while ((*neighbor_voxel_index)(i) < 0) {
       (*neighbor_block_index)(i)--;
       (*neighbor_voxel_index)(i) += voxels_per_side;
     }
     while ((*neighbor_voxel_index)(i) >=
            static_cast<IndexElement>(voxels_per_side)) {
       (*neighbor_block_index)(i)++;
       (*neighbor_voxel_index)(i) -= voxels_per_side;
     }
   }
 }

 static void getFromBlockAndVoxelIndex(
     const BlockIndex& block_index, const VoxelIndex& voxel_index,
     const size_t voxels_per_side, AlignedVector<VoxelKey>* neighbors_ptr) {
   CHECK_NOTNULL(neighbors_ptr)->resize(kConnectivity);

   AlignedVector<VoxelKey>& neighbors = *neighbors_ptr;
   for (unsigned int i = 0u; i < kConnectivity; ++i) {
     VoxelKey& neighbor = neighbors[i];
     getFromBlockAndVoxelIndexAndDirection(block_index, voxel_index,
                                           kOffsets.col(i), voxels_per_side,
                                           &neighbor.first, &neighbor.second);
   }
 }

 static SignedIndex getOffsetBetweenVoxels(const BlockIndex& start_block_index,
                                           const VoxelIndex& start_voxel_index,
                                           const BlockIndex& end_block_index,
                                           const VoxelIndex& end_voxel_index,
                                           const size_t voxels_per_side) {
   CHECK_NE(voxels_per_side, 0u);
   return (end_voxel_index - start_voxel_index) +
          (end_block_index - start_block_index) * voxels_per_side;
```

(continues on next page)

(continued from previous page)

```
  }
};
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_NEIGHBOR_TOOLS_H_
```

Includes

- `voxblox/core/common.h` (*File common.h*)
- `voxblox/core/layer.h` (*File layer.h*)

Included By

- *File esdf_integrator.h*

Namespaces

- *Namespace voxblox*

Classes

- *Template Class Neighborhood*
- *Class NeighborhoodLookupTables*

File objects.h

Contents

- *Definition (`voxblox/include/voxblox/simulation/objects.h`)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/simulation/objects.h`)

Program Listing for File objects.h

Return to documentation for file (`voxblox/include/voxblox/simulation/objects.h`)

```
#ifndef VOXBLOX_SIMULATION_OBJECTS_H_
#define VOXBLOX_SIMULATION_OBJECTS_H_

#include <algorithm>
#include <iostream>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

// Heavily inspired by @mfehr's OccupancyGridGenerator.
namespace voxblox {

// Should this be full virtual?
class Object {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  // A wall is an infinite plane.
  enum Type { kSphere = 0, kCube, kPlane, kCylinder };

  Object(const Point& center, Type type)
      : Object(center, type, Color::White()) {}
  Object(const Point& center, Type type, const Color& color)
      : center_(center), type_(type), color_(color) {}
  virtual ~Object() {}

  virtual FloatingPoint getDistanceToPoint(const Point& point) const = 0;

  Color getColor() const { return color_; }

  virtual bool getRayIntersection(const Point& ray_origin,
                                  const Point& ray_direction,
                                  FloatingPoint max_dist,
                                  Point* intersect_point,
                                  FloatingPoint* intersect_dist) const = 0;

 protected:
  Point center_;
  Type type_;
  Color color_;
};

class Sphere : public Object {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  Sphere(const Point& center, FloatingPoint radius)
      : Object(center, Type::kSphere), radius_(radius) {}
  Sphere(const Point& center, FloatingPoint radius, const Color& color)
      : Object(center, Type::kSphere, color), radius_(radius) {}

  virtual FloatingPoint getDistanceToPoint(const Point& point) const {
    FloatingPoint distance = (center_ - point).norm() - radius_;
    return distance;
  }
```

(continues on next page)

(continued from previous page)

```
  virtual bool getRayIntersection(const Point& ray_origin,
                                  const Point& ray_direction,
                                  FloatingPoint max_dist,
                                  Point* intersect_point,
                                  FloatingPoint* intersect_dist) const {
    // From https://en.wikipedia.org/wiki/Line%E2%80%93sphere_intersection
    // x = o + dl is the ray equation
    // r = sphere radius, c = sphere center
    FloatingPoint under_square_root =
        pow(ray_direction.dot(ray_origin - center_), 2) -
        (ray_origin - center_).squaredNorm() + pow(radius_, 2);

    // No real roots = no intersection.
    if (under_square_root < 0.0) {
      return false;
    }

    FloatingPoint d =
        -(ray_direction.dot(ray_origin - center_)) - sqrt(under_square_root);

    // Intersection behind the origin.
    if (d < 0.0) {
      return false;
    }
    // Intersection greater than max dist, so no intersection in the sensor
    // range.
    if (d > max_dist) {
      return false;
    }

    *intersect_point = ray_origin + d * ray_direction;
    *intersect_dist = d;
    return true;
  }

 protected:
  FloatingPoint radius_;
};

class Cube : public Object {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  Cube(const Point& center, const Point& size)
      : Object(center, Type::kCube), size_(size) {}
  Cube(const Point& center, const Point& size, const Color& color)
      : Object(center, Type::kCube, color), size_(size) {}

  virtual FloatingPoint getDistanceToPoint(const Point& point) const {
    // Solution from http://stackoverflow.com/questions/5254838/
    // calculating-distance-between-a-point-and-a-rectangular-box-nearest-point

    Point distance_vector = Point::Zero();
    distance_vector.x() =
        std::max(std::max(center_.x() - size_.x() / 2.0 - point.x(), 0.0),
                 point.x() - center_.x() - size_.x() / 2.0);
    distance_vector.y() =
```

(continues on next page)

(continued from previous page)

```
        std::max(std::max(center_.y() - size_.y() / 2.0 - point.y(), 0.0),
                 point.y() - center_.y() - size_.y() / 2.0);
    distance_vector.z() =
        std::max(std::max(center_.z() - size_.z() / 2.0 - point.z(), 0.0),
                 point.z() - center_.z() - size_.z() / 2.0);

    FloatingPoint distance = distance_vector.norm();

    // Basically 0... Means it's inside!
    if (distance < kEpsilon) {
      distance_vector.x() = std::max(center_.x() - size_.x() / 2.0 - point.x(),
                                     point.x() - center_.x() - size_.x() / 2.0);
      distance_vector.y() = std::max(center_.y() - size_.y() / 2.0 - point.y(),
                                     point.y() - center_.y() - size_.y() / 2.0);
      distance_vector.z() = std::max(center_.z() - size_.z() / 2.0 - point.z(),
                                     point.z() - center_.z() - size_.z() / 2.0);
      distance = distance_vector.maxCoeff();
    }

    return distance;
  }

  virtual bool getRayIntersection(const Point& ray_origin,
                                  const Point& ray_direction,
                                  FloatingPoint max_dist,
                                  Point* intersect_point,
                                  FloatingPoint* intersect_dist) const {
    // Adapted from https://www.scratchapixel.com/lessons/3d-basic-rendering/
    // minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection
    // Compute min and max limits in 3D.

    // Precalculate signs and inverse directions.
    Point inv_dir(1.0 / ray_direction.x(), 1.0 / ray_direction.y(),
                  1.0 / ray_direction.z());
    Eigen::Vector3i ray_sign(inv_dir.x() < 0.0, inv_dir.y() < 0.0,
                             inv_dir.z() < 0.0);

    Point bounds[2];
    bounds[0] = center_ - size_ / 2.0;
    bounds[1] = center_ + size_ / 2.0;

    FloatingPoint tmin =
        (bounds[ray_sign.x()].x() - ray_origin.x()) * inv_dir.x();
    FloatingPoint tmax =
        (bounds[1 - ray_sign.x()].x() - ray_origin.x()) * inv_dir.x();
    FloatingPoint tymin =
        (bounds[ray_sign.y()].y() - ray_origin.y()) * inv_dir.y();
    FloatingPoint tymax =
        (bounds[1 - ray_sign.y()].y() - ray_origin.y()) * inv_dir.y();

    if ((tmin > tymax) || (tymin > tmax)) return false;
    if (tymin > tmin) tmin = tymin;
    if (tymax < tmax) tmax = tymax;

    FloatingPoint tzmin =
        (bounds[ray_sign.z()].z() - ray_origin.z()) * inv_dir.z();
    FloatingPoint tzmax =
```

(continues on next page)

(continued from previous page)

```
        (bounds[1 - ray_sign.z()].z() - ray_origin.z()) * inv_dir.z();

    if ((tmin > tzmax) || (tzmin > tmax)) return false;
    if (tzmin > tmin) tmin = tzmin;
    if (tzmax < tmax) tmax = tzmax;

    FloatingPoint t = tmin;
    if (t < 0.0) {
      t = tmax;
      if (t < 0.0) {
        return false;
      }
    }

    if (t > max_dist) {
      return false;
    }

    *intersect_dist = t;
    *intersect_point = ray_origin + ray_direction * t;

    return true;
  }

 protected:
  Point size_;
};

class PlaneObject : public Object {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  PlaneObject(const Point& center, const Point& normal)
      : Object(center, Type::kPlane), normal_(normal) {}
  PlaneObject(const Point& center, const Point& normal, const Color& color)
      : Object(center, Type::kPlane, color), normal_(normal) {
    CHECK_NEAR(normal.norm(), 1.0, 1e-3);
  }

  virtual FloatingPoint getDistanceToPoint(const Point& point) const {
    // Compute the 'd' in ax + by + cz + d = 0:
    // This is actually the scalar product I guess.
    FloatingPoint d = -normal_.dot(center_);
    FloatingPoint p = d / normal_.norm();

    FloatingPoint distance = normal_.dot(point) + p;
    return distance;
  }

  virtual bool getRayIntersection(const Point& ray_origin,
                                  const Point& ray_direction,
                                  FloatingPoint max_dist,
                                  Point* intersect_point,
                                  FloatingPoint* intersect_dist) const {
    // From https://en.wikipedia.org/wiki/Line%E2%80%93plane_intersection
    // Following notation of sphere more...
    // x = o + dl is the ray equation
```

(continues on next page)

(continued from previous page)

```
    // n = normal, c = plane 'origin'
    FloatingPoint denominator = ray_direction.dot(normal_);
    if (std::abs(denominator) < kEpsilon) {
      // Lines are parallel, no intersection.
      return false;
    }
    FloatingPoint d = (center_ - ray_origin).dot(normal_) / denominator;
    if (d < 0.0) {
      return false;
    }
    if (d > max_dist) {
      return false;
    }
    *intersect_point = ray_origin + d * ray_direction;
    *intersect_dist = d;
    return true;
  }

 protected:
  Point normal_;
};

class Cylinder : public Object {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  Cylinder(const Point& center, FloatingPoint radius, FloatingPoint height)
      : Object(center, Type::kCylinder), radius_(radius), height_(height) {}
  Cylinder(const Point& center, FloatingPoint radius, FloatingPoint height,
           const Color& color)
      : Object(center, Type::kCylinder, color),
        radius_(radius),
        height_(height) {}

  virtual FloatingPoint getDistanceToPoint(const Point& point) const {
    // From: https://math.stackexchange.com/questions/2064745/
    // 3 cases, depending on z of point.
    // First case: in plane with the cylinder. This also takes care of inside
    // case.
    FloatingPoint distance = 0.0;

    FloatingPoint min_z_limit = center_.z() - height_ / 2.0;
    FloatingPoint max_z_limit = center_.z() + height_ / 2.0;
    if (point.z() >= min_z_limit && point.z() <= max_z_limit) {
      distance = (point.head<2>() - center_.head<2>()).norm() - radius_;
    } else if (point.z() > max_z_limit) {
      // Case 2: above the cylinder.
      distance = std::sqrt(
          std::max((point.head<2>() - center_.head<2>()).squaredNorm() -
                       radius_ * radius_,
                   static_cast<FloatingPoint>(0.0)) +
          (point.z() - max_z_limit) * (point.z() - max_z_limit));
    } else {
      // Case 3: below cylinder.
      distance = std::sqrt(
          std::max((point.head<2>() - center_.head<2>()).squaredNorm() -
                       radius_ * radius_,
```

(continues on next page)

(continued from previous page)

```
                static_cast<FloatingPoint>(0.0)) +
        (point.z() - min_z_limit) * (point.z() - min_z_limit));
  }
  return distance;
}

virtual bool getRayIntersection(const Point& ray_origin,
                                const Point& ray_direction,
                                FloatingPoint max_dist,
                                Point* intersect_point,
                                FloatingPoint* intersect_dist) const {
  // From http://woo4.me/wootracer/cylinder-intersection/
  // and http://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html
  // Define ray as P = E + tD, where E is ray_origin and D is ray_direction.
  // We define our cylinder as centered in the xy coordinate system, so
  // E in this case is actually ray_origin - center_.
  Point vector_E = ray_origin - center_;
  Point vector_D = ray_direction;  // Axis aligned.

  FloatingPoint a = vector_D.x() * vector_D.x() + vector_D.y() * vector_D.y();
  FloatingPoint b =
      2 * vector_E.x() * vector_D.x() + 2 * vector_E.y() * vector_D.y();
  FloatingPoint c = vector_E.x() * vector_E.x() +
                    vector_E.y() * vector_E.y() - radius_ * radius_;

  // t = (-b +- sqrt(b^2 - 4ac))/2a
  // t only has solutions if b^2 - 4ac >= 0
  FloatingPoint t1 = -1.0;
  FloatingPoint t2 = -1.0;

  // Make sure we don't divide by 0.
  if (std::abs(a) < kEpsilon) {
    return false;
  }

  FloatingPoint under_square_root = b * b - 4 * a * c;
  if (under_square_root < 0) {
    return false;
  }
  if (under_square_root <= kEpsilon) {
    t1 = -b / (2 * a);
    // Just keep t2 at invalid default value.
  } else {
    // 2 ts.
    t1 = (-b + std::sqrt(under_square_root)) / (2 * a);
    t2 = (-b - std::sqrt(under_square_root)) / (2 * a);
  }

  // Great, now we got some ts, time to figure out whether we hit the cylinder
  // or the endcaps.
  FloatingPoint t = max_dist;

  FloatingPoint z1 = vector_E.z() + t1 * vector_D.z();
  FloatingPoint z2 = vector_E.z() + t2 * vector_D.z();
  bool t1_valid = t1 >= 0.0 && z1 >= -height_ / 2.0 && z1 <= height_ / 2.0;
  bool t2_valid = t2 >= 0.0 && z2 >= -height_ / 2.0 && z2 <= height_ / 2.0;
```

(continues on next page)

(continued from previous page)

```
    // Get the endcaps and their validity.
    // Check end-cap intersections now... :(
    FloatingPoint t3, t4;
    bool t3_valid = false, t4_valid = false;

    // Make sure we don't divide by 0.
    if (std::abs(vector_D.z()) > kEpsilon) {
      // t3 is the bottom end-cap, t4 is the top.
      t3 = (-height_ / 2.0 - vector_E.z()) / vector_D.z();
      t4 = (height_ / 2.0 - vector_E.z()) / vector_D.z();

      Point q3 = vector_E + t3 * vector_D;
      Point q4 = vector_E + t4 * vector_D;

      t3_valid = t3 >= 0.0 && q3.head<2>().norm() < radius_;
      t4_valid = t4 >= 0.0 && q4.head<2>().norm() < radius_;
    }

    if (!(t1_valid || t2_valid || t3_valid || t4_valid)) {
      return false;
    }
    if (t1_valid) {
      t = std::min(t, t1);
    }
    if (t2_valid) {
      t = std::min(t, t2);
    }
    if (t3_valid) {
      t = std::min(t, t3);
    }
    if (t4_valid) {
      t = std::min(t, t4);
    }

    // Intersection greater than max dist, so no intersection in the sensor
    // range.
    if (t >= max_dist) {
      return false;
    }

    // Back to normal coordinates now.
    *intersect_point = ray_origin + t * ray_direction;
    *intersect_dist = t;
    return true;
  }

 protected:
  FloatingPoint radius_;
  FloatingPoint height_;
};

}  // namespace voxblox

#endif  // VOXBLOX_SIMULATION_OBJECTS_H_
```

Includes

- `algorithm`
- `iostream`
- `voxblox/core/common.h` (*File common.h*)
- `voxblox/core/layer.h` (*File layer.h*)
- `voxblox/core/voxel.h` (*File voxel.h*)

Included By

- *File simulation_world.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class Cube*
- *Class Cylinder*
- *Class Object*
- *Class PlaneObject*
- *Class Sphere*

File occupancy_integrator.h

Contents

- *Definition (`voxblox/include/voxblox/integrator/occupancy_integrator.h`)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/integrator/occupancy_integrator.h`)

Program Listing for File occupancy_integrator.h

Return to documentation for file (`voxblox/include/voxblox/integrator/occupancy_integrator.h`)

```
#ifndef VOXBLOX_INTEGRATOR_OCCUPANCY_INTEGRATOR_H_
#define VOXBLOX_INTEGRATOR_OCCUPANCY_INTEGRATOR_H_

#include <algorithm>
#include <vector>

#include <glog/logging.h>
#include <Eigen/Core>

#include "voxblox/core/block_hash.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/utils/timing.h"

namespace voxblox {
class OccupancyIntegrator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    float probability_hit = 0.65f;
    float probability_miss = 0.4f;
    float threshold_min = 0.12f;
    float threshold_max = 0.97f;
    float threshold_occupancy = 0.7f;
    FloatingPoint min_ray_length_m = 0.1;
    FloatingPoint max_ray_length_m = 5.0;
  };

  OccupancyIntegrator(const Config& config, Layer<OccupancyVoxel>* layer)
      : config_(config), layer_(layer) {
    DCHECK(layer_ != NULL);
    DCHECK_GT(layer_->voxel_size(), 0.0);
    DCHECK_GT(layer_->block_size(), 0.0);
    DCHECK_GT(layer_->voxels_per_side(), 0u);

    voxel_size_ = layer_->voxel_size();
    block_size_ = layer_->block_size();
    voxels_per_side_ = layer_->voxels_per_side();

    voxel_size_inv_ = 1.0 / voxel_size_;
    block_size_inv_ = 1.0 / block_size_;
    voxels_per_side_inv_ = 1.0 / voxels_per_side_;

    // Cache rest of the probability settings.
    prob_hit_log_ = logOddsFromProbability(config_.probability_hit);
    prob_miss_log_ = logOddsFromProbability(config_.probability_miss);
    clamp_min_log_ = logOddsFromProbability(config_.threshold_min);
    clamp_max_log_ = logOddsFromProbability(config_.threshold_max);
    min_occupancy_log_ = logOddsFromProbability(config_.threshold_occupancy);
  }

  inline void updateOccupancyVoxel(bool occupied, OccupancyVoxel* occ_voxel) {
    DCHECK(occ_voxel != NULL);
```

(continues on next page)

(continued from previous page)

```
    // Set voxel to observed.
    occ_voxel->observed = true;
    // Skip update if necessary.
    float log_odds_update = occupied ? prob_hit_log_ : prob_miss_log_;
    if ((log_odds_update >= 0 &&
         occ_voxel->probability_log >= clamp_max_log_) ||
        (log_odds_update <= 0 &&
         occ_voxel->probability_log <= clamp_min_log_)) {
      return;
    }
    // Otherwise update and reclamp.
    occ_voxel->probability_log = std::min(
        std::max(occ_voxel->probability_log + log_odds_update, clamp_min_log_),
        clamp_max_log_);
  }

  void integratePointCloud(const Transformation& T_G_C,
                           const Pointcloud& points_C) {
    timing::Timer integrate_timer("integrate_occ");

    const Point& origin = T_G_C.getPosition();

    LongIndexSet free_cells;
    LongIndexSet occupied_cells;

    const Point start_scaled = origin * voxel_size_inv_;
    Point end_scaled = Point::Zero();

    for (size_t pt_idx = 0; pt_idx < points_C.size(); ++pt_idx) {
      const Point& point_C = points_C[pt_idx];
      const Point point_G = T_G_C * point_C;
      const Ray unit_ray = (point_G - origin).normalized();

      timing::Timer cast_ray_timer("integrate_occ/cast_ray");

      AlignedVector<GlobalIndex> global_voxel_indices;
      FloatingPoint ray_distance = (point_G - origin).norm();
      if (ray_distance < config_.min_ray_length_m) {
        continue;
      } else if (ray_distance > config_.max_ray_length_m) {
        // Simply clear up until the max ray distance in this case.
        end_scaled =
            (origin + config_.max_ray_length_m * unit_ray) * voxel_size_inv_;

        if (free_cells.find(getGridIndexFromPoint<GlobalIndex>(end_scaled)) ==
            free_cells.end()) {
          castRay(start_scaled, end_scaled, &global_voxel_indices);
          free_cells.insert(global_voxel_indices.begin(),
                            global_voxel_indices.end());
        }
      } else {
        end_scaled = point_G * voxel_size_inv_;
        if (occupied_cells.find(getGridIndexFromPoint<GlobalIndex>(
                end_scaled)) == occupied_cells.end()) {
          castRay(start_scaled, end_scaled, &global_voxel_indices);

          if (global_voxel_indices.size() > 2) {
```

(continues on next page)

(continued from previous page)

```
          free_cells.insert(global_voxel_indices.begin(),
                            global_voxel_indices.end() - 1);
          occupied_cells.insert(global_voxel_indices.back());
        }
      }
    }
    cast_ray_timer.Stop();
  }

  timing::Timer update_voxels_timer("integrate_occ/update_occupancy");

  // Clean up the lists: remove any occupied cells from free cells.
  for (const GlobalIndex& global_index : occupied_cells) {
    LongIndexSet::iterator cell_it = free_cells.find(global_index);
    if (cell_it != free_cells.end()) {
      free_cells.erase(cell_it);
    }
  }
  // Then actually update the occupancy voxels.
  BlockIndex last_block_idx = BlockIndex::Zero();
  Block<OccupancyVoxel>::Ptr block;

  bool occupied = false;
  for (const GlobalIndex& global_voxel_idx : free_cells) {
    BlockIndex block_idx = getBlockIndexFromGlobalVoxelIndex(
        global_voxel_idx, voxels_per_side_inv_);
    VoxelIndex local_voxel_idx =
        getLocalFromGlobalVoxelIndex(global_voxel_idx, voxels_per_side_);

    if (!block || block_idx != last_block_idx) {
      block = layer_->allocateBlockPtrByIndex(block_idx);
      block->updated() = true;
      last_block_idx = block_idx;
    }

    OccupancyVoxel& occ_voxel = block->getVoxelByVoxelIndex(local_voxel_idx);
    updateOccupancyVoxel(occupied, &occ_voxel);
  }
  block.reset();

  occupied = true;
  for (const GlobalIndex& global_voxel_idx : occupied_cells) {
    BlockIndex block_idx = getBlockIndexFromGlobalVoxelIndex(
        global_voxel_idx, voxels_per_side_inv_);
    VoxelIndex local_voxel_idx =
        getLocalFromGlobalVoxelIndex(global_voxel_idx, voxels_per_side_);

    if (!block || block_idx != last_block_idx) {
      block = layer_->allocateBlockPtrByIndex(block_idx);
      block->updated() = true;
      last_block_idx = block_idx;
    }

    OccupancyVoxel& occ_voxel = block->getVoxelByVoxelIndex(local_voxel_idx);
    updateOccupancyVoxel(occupied, &occ_voxel);
  }
```

(continues on next page)

(continued from previous page)

```
    update_voxels_timer.Stop();
    integrate_timer.Stop();
  }

 protected:
  Config config_;

  Layer<OccupancyVoxel>* layer_;

  // Cached map config.
  FloatingPoint voxel_size_;
  size_t voxels_per_side_;
  FloatingPoint block_size_;

  float prob_hit_log_;
  float prob_miss_log_;
  float clamp_min_log_;
  float clamp_max_log_;
  float min_occupancy_log_;

  // Derived types.
  FloatingPoint voxel_size_inv_;
  FloatingPoint voxels_per_side_inv_;
  FloatingPoint block_size_inv_;
};

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_OCCUPANCY_INTEGRATOR_H_
```

Includes

- Eigen/Core
- algorithm
- glog/logging.h
- vector
- voxblox/core/block_hash.h (*File block_hash.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/integrator_utils.h (*File integrator_utils.h*)
- voxblox/utils/timing.h (*File timing.h*)

Included By

- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct OccupancyIntegrator::Config*
- *Class OccupancyIntegrator*

File occupancy_map.h

Contents

- *Definition (voxblox/include/voxblox/core/occupancy_map.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/core/occupancy_map.h)

Program Listing for File occupancy_map.h

Return to documentation for file (voxblox/include/voxblox/core/occupancy_map.h)

```
#ifndef VOXBLOX_CORE_OCCUPANCY_MAP_H_
#define VOXBLOX_CORE_OCCUPANCY_MAP_H_

#include <glog/logging.h>
#include <memory>
#include <utility>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
class OccupancyMap {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::shared_ptr<OccupancyMap> Ptr;

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    FloatingPoint occupancy_voxel_size = 0.2;
    size_t occupancy_voxels_per_side = 16u;
  };
```

(continues on next page)

(continued from previous page)

```
  explicit OccupancyMap(const Config& config)
      : occupancy_layer_(new Layer<OccupancyVoxel>(
            config.occupancy_voxel_size, config.occupancy_voxels_per_side)) {
    block_size_ =
        config.occupancy_voxel_size * config.occupancy_voxels_per_side;
  }

  // Creates a new OccupancyMap based on a COPY of this layer.
  explicit OccupancyMap(const Layer<OccupancyVoxel>& layer)
      : OccupancyMap(aligned_shared<Layer<OccupancyVoxel>>(layer)) {}

  // Creates a new OccupancyMap that contains this layer.
  explicit OccupancyMap(Layer<OccupancyVoxel>::Ptr layer)
      : occupancy_layer_(layer) {
    CHECK(layer);
    block_size_ = layer->block_size();
  }

  virtual ~OccupancyMap() {}

  Layer<OccupancyVoxel>* getOccupancyLayerPtr() {
    return occupancy_layer_.get();
  }
  const Layer<OccupancyVoxel>& getOccupancyLayer() const {
    return *occupancy_layer_;
  }

  FloatingPoint block_size() const { return block_size_; }

 protected:
  FloatingPoint block_size_;

  // The layers.
  Layer<OccupancyVoxel>::Ptr occupancy_layer_;
};

}  // namespace voxblox

#endif  // VOXBLOX_CORE_OCCUPANCY_MAP_H_
```

Includes

- glog/logging.h
- memory
- utility
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Namespaces

- *Namespace voxblox*

Classes

- *Struct OccupancyMap::Config*
- *Class OccupancyMap*

File planning_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/planning_utils.h)*
- *Includes*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/planning_utils.h)

Program Listing for File planning_utils.h

Return to documentation for file (voxblox/include/voxblox/utils/planning_utils.h)

```
#ifndef VOXBLOX_UTILS_PLANNING_UTILS_H_
#define VOXBLOX_UTILS_PLANNING_UTILS_H_

#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
namespace utils {

template <typename VoxelType>
void getSphereAroundPoint(const Layer<VoxelType>& layer, const Point& center,
                          FloatingPoint radius,
                          HierarchicalIndexMap* block_voxel_list);

template <typename VoxelType>
void getAndAllocateSphereAroundPoint(const Point& center, FloatingPoint radius,
                                     Layer<VoxelType>* layer,
                                     HierarchicalIndexMap* block_voxel_list);

template <typename VoxelType>
void fillSphereAroundPoint(const Point& center, const FloatingPoint radius,
                           const FloatingPoint max_distance_m,
                           Layer<VoxelType>* layer);
template <typename VoxelType>
void clearSphereAroundPoint(const Point& center, const FloatingPoint radius,
                            const FloatingPoint max_distance_m,
```

(continues on next page)

(continued from previous page)

```
                              Layer<VoxelType>* layer);

template <typename VoxelType>
void computeMapBoundsFromLayer(const voxblox::Layer<VoxelType>& layer,
                               Eigen::Vector3d* lower_bound,
                               Eigen::Vector3d* upper_bound);

}  // namespace utils
}  // namespace voxblox

#include "voxblox/utils/planning_utils_inl.h"

#endif  // VOXBLOX_UTILS_PLANNING_UTILS_H_
```

Includes

- `voxblox/core/layer.h` (*File layer.h*)
- `voxblox/core/voxel.h` (*File voxel.h*)
- `voxblox/utils/planning_utils_inl.h` (*File planning_utils_inl.h*)

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::utils*

File planning_utils_inl.h

Contents

- *Definition (voxblox/include/voxblox/utils/planning_utils_inl.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/planning_utils_inl.h)

Program Listing for File planning_utils_inl.h

Return to documentation for file (`voxblox/include/voxblox/utils/planning_utils_inl.h`)

```
#ifndef VOXBLOX_UTILS_PLANNING_UTILS_INL_H_
#define VOXBLOX_UTILS_PLANNING_UTILS_INL_H_

#include <algorithm>
```

(continues on next page)

(continued from previous page)

```
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
namespace utils {

template <typename VoxelType>
void getSphereAroundPoint(const Layer<VoxelType>& layer, const Point& center,
                          FloatingPoint radius,
                          HierarchicalIndexMap* block_voxel_list) {
  CHECK_NOTNULL(block_voxel_list);
  float voxel_size = layer.voxel_size();
  float voxel_size_inv = 1.0 / layer.voxel_size();
  int voxels_per_side = layer.voxels_per_side();

  const GlobalIndex center_index =
      getGridIndexFromPoint<GlobalIndex>(center, voxel_size_inv);
  const FloatingPoint radius_in_voxels = radius / voxel_size;

  for (FloatingPoint x = -radius_in_voxels; x <= radius_in_voxels; x++) {
    for (FloatingPoint y = -radius_in_voxels; y <= radius_in_voxels; y++) {
      for (FloatingPoint z = -radius_in_voxels; z <= radius_in_voxels; z++) {
        Point point_voxel_space(x, y, z);

        // check if point is inside the spheres radius
        if (point_voxel_space.norm() <= radius_in_voxels) {
          GlobalIndex voxel_offset_index =
              Eigen::floor(point_voxel_space.array()).cast<LongIndexElement>();

          // Get the block and voxel indices from this.
          BlockIndex block_index;
          VoxelIndex voxel_index;

          getBlockAndVoxelIndexFromGlobalVoxelIndex(
              voxel_offset_index + center_index, voxels_per_side, &block_index,
              &voxel_index);
          (*block_voxel_list)[block_index].push_back(voxel_index);
        }
      }
    }
  }
}

template <typename VoxelType>
void getAndAllocateSphereAroundPoint(const Point& center, FloatingPoint radius,
                                     Layer<VoxelType>* layer,
                                     HierarchicalIndexMap* block_voxel_list) {
  CHECK_NOTNULL(layer);
  CHECK_NOTNULL(block_voxel_list);
  getSphereAroundPoint(*layer, center, radius, block_voxel_list);
  for (auto it = block_voxel_list->begin(); it != block_voxel_list->end();
       ++it) {
    layer->allocateBlockPtrByIndex(it->first);
  }
}
```

(continues on next page)

(continued from previous page)

```
// This function sets all voxels within a Euclidean distance of the center
// to a value equal to the distance of the point from the center, essentially
// making it a filled sphere with that center and radius.
// Marks the new points as hallucinated and fixed.
template <typename VoxelType>
void fillSphereAroundPoint(const Point& center, const FloatingPoint radius,
                           const FloatingPoint max_distance_m,
                           Layer<VoxelType>* layer) {
  CHECK_NOTNULL(layer);
  HierarchicalIndexMap block_voxel_list;
  getAndAllocateSphereAroundPoint(center, radius, layer, &block_voxel_list);

  for (auto it = block_voxel_list.begin(); it != block_voxel_list.end(); ++it) {
    typename Block<VoxelType>::Ptr block_ptr =
        layer->getBlockPtrByIndex(it->first);
    for (const VoxelIndex& voxel_index : it->second) {
      Point point = block_ptr->computeCoordinatesFromVoxelIndex(voxel_index);
      Point voxel_center_vec = point - center;

      VoxelType& voxel = block_ptr->getVoxelByVoxelIndex(voxel_index);
      // The distance of the voxel should map to actual meaningful
      // Euclidean distance; in this case, the sphere center has the max
      // distance, and the edges should have the lowest distance.
      // Also compress this to the max distance given.
      const FloatingPoint new_distance =
          std::max(voxel_center_vec.norm() - radius, -max_distance_m);

      if (!voxel.observed || new_distance < voxel.distance) {
        voxel.distance = new_distance;
        voxel.observed = true;
        voxel.hallucinated = true;
        voxel.fixed = true;
        block_ptr->updated() = true;
        block_ptr->has_data() = true;
      }
    }
  }
}

// Similar to above, clears the area around the specified point, marking it as
// hallucinated and fixed.
template <typename VoxelType>
void clearSphereAroundPoint(const Point& center, const FloatingPoint radius,
                            const FloatingPoint max_distance_m,
                            Layer<VoxelType>* layer) {
  CHECK_NOTNULL(layer);
  HierarchicalIndexMap block_voxel_list;
  getAndAllocateSphereAroundPoint(center, radius, layer, &block_voxel_list);

  for (auto it = block_voxel_list.begin(); it != block_voxel_list.end(); ++it) {
    typename Block<VoxelType>::Ptr block_ptr =
        layer->getBlockPtrByIndex(it->first);
    for (const VoxelIndex& voxel_index : it->second) {
      Point point = block_ptr->computeCoordinatesFromVoxelIndex(voxel_index);
      Point voxel_center_vec = point - center;

      VoxelType& voxel = block_ptr->getVoxelByVoxelIndex(voxel_index);
```

(continues on next page)

(continued from previous page)

```
      // How far is voxel from edge of free sphere. The values should be
      // biggest in the center, smallest outside.
      const FloatingPoint new_distance =
          std::min(radius - voxel_center_vec.norm(), max_distance_m);

      if (!voxel.observed || new_distance > voxel.distance) {
        voxel.distance = new_distance;
        voxel.observed = true;
        voxel.hallucinated = true;
        voxel.fixed = true;
        block_ptr->updated() = true;
        block_ptr->has_data() = true;
      }
    }
  }
}

// Utility function to get map bounds from an arbitrary layer.
template <typename VoxelType>
void computeMapBoundsFromLayer(const voxblox::Layer<VoxelType>& layer,
                               Eigen::Vector3d* lower_bound,
                               Eigen::Vector3d* upper_bound) {
  FloatingPoint block_size = layer.block_size();

  BlockIndexList all_blocks;
  layer.getAllAllocatedBlocks(&all_blocks);

  BlockIndex lower_bound_index;
  BlockIndex upper_bound_index;

  bool first_block = true;

  for (const voxblox::BlockIndex& block_index : all_blocks) {
    if (first_block) {
      lower_bound_index = block_index;
      upper_bound_index = block_index;
      first_block = false;
      continue;
    }

    lower_bound_index = lower_bound_index.array().min(block_index.array());
    upper_bound_index = upper_bound_index.array().max(block_index.array());
  }

  *lower_bound =
      getOriginPointFromGridIndex(lower_bound_index, block_size).cast<double>();
  *upper_bound =
      (getOriginPointFromGridIndex(upper_bound_index, block_size).array() +
       block_size)
          .cast<double>();
}

}  // namespace utils
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_PLANNING_UTILS_INL_H_
```

Includes

- `algorithm`
- `voxblox/core/layer.h` (*File layer.h*)
- `voxblox/core/voxel.h` (*File voxel.h*)

Included By

- *File planning_utils.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::utils*

File ply_writer.h

Contents

- *Definition (`voxblox/include/voxblox/io/ply_writer.h`)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/io/ply_writer.h`)

Program Listing for File ply_writer.h

Return to documentation for file (`voxblox/include/voxblox/io/ply_writer.h`)

```
#ifndef VOXBLOX_IO_PLY_WRITER_H_
#define VOXBLOX_IO_PLY_WRITER_H_

#include <fstream>  // NOLINT
#include <string>

#include <glog/logging.h>

#include "voxblox/core/common.h"

namespace voxblox {

namespace io {
class PlyWriter {
 public:
```

(continues on next page)

(continued from previous page)

```
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  explicit PlyWriter(const std::string& filename)
      : header_written_(false),
        parameters_set_(false),
        vertices_total_(0),
        vertices_written_(0),
        has_color_(false),
        file_(filename) {}

  virtual ~PlyWriter() {}

  void addVerticesWithProperties(size_t num_vertices, bool has_color) {
    vertices_total_ = num_vertices;
    has_color_ = has_color;
    parameters_set_ = true;
  }

  bool writeHeader() {
    if (!file_) {
      // Output a warning -- couldn't open file?
      LOG(WARNING) << "Could not open file for PLY output.";
      return false;
    }
    if (!parameters_set_) {
      LOG(WARNING) << "Could not write header out -- parameters not set.";
      return false;
    }
    if (header_written_) {
      LOG(WARNING) << "Header already written.";
      return false;
    }

    file_ << "ply" << std::endl;
    file_ << "format ascii 1.0" << std::endl;
    file_ << "element vertex " << vertices_total_ << std::endl;
    file_ << "property float x" << std::endl;
    file_ << "property float y" << std::endl;
    file_ << "property float z" << std::endl;

    if (has_color_) {
      file_ << "property uchar red" << std::endl;
      file_ << "property uchar green" << std::endl;
      file_ << "property uchar blue" << std::endl;
    }

    file_ << "end_header" << std::endl;

    header_written_ = true;
    return true;
  }

  bool writeVertex(const Point& coord) {
    if (!header_written_) {
      if (!writeHeader()) {
        return false;
      }
```

(continues on next page)

(continued from previous page)

```
    }
    if (vertices_written_ >= vertices_total_ || has_color_) {
      return false;
    }
    file_ << coord.x() << " " << coord.y() << " " << coord.z() << std::endl;
    return true;
  }

  bool writeVertex(const Point& coord, const Color& rgb) {
    if (!header_written_) {
      if (!writeHeader()) {
        return false;
      }
    }
    if (vertices_written_ >= vertices_total_ || !has_color_) {
      return false;
    }
    file_ << coord.x() << " " << coord.y() << " " << coord.z() << " ";
    file_ << static_cast<int>(rgb.r) << " " << static_cast<int>(rgb.g) << " "
          << static_cast<int>(rgb.b) << std::endl;
    return true;
  }

  void closeFile() { file_.close(); }

 private:
  bool header_written_;
  bool parameters_set_;

  size_t vertices_total_;
  size_t vertices_written_;
  bool has_color_;

  std::ofstream file_;
};

}  // namespace io

}  // namespace voxblox

#endif  // VOXBLOX_IO_PLY_WRITER_H_
```

Includes

- fstream
- glog/logging.h
- string
- voxblox/core/common.h (*File common.h*)

Namespaces

- *Namespace voxblox*

- *Namespace voxblox::io*

Classes

- *Class PlyWriter*

File protobuf_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/protobuf_utils.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/protobuf_utils.h)

Program Listing for File protobuf_utils.h

Return to documentation for file (voxblox/include/voxblox/utils/protobuf_utils.h)

```
#ifndef VOXBLOX_UTILS_PROTOBUF_UTILS_H_
#define VOXBLOX_UTILS_PROTOBUF_UTILS_H_

#include <fstream>
#include <glog/logging.h>
#include <google/protobuf/message.h>
#include <google/protobuf/message_lite.h>

namespace voxblox {

namespace utils {
bool readProtoMsgCountToStream(std::fstream* stream_in, uint32_t* message_count,
                               uint32_t* byte_offset);

bool writeProtoMsgCountToStream(uint32_t message_count,
                                std::fstream* stream_out);

bool readProtoMsgFromStream(std::fstream* stream_in,
                            google::protobuf::Message* message,
                            uint32_t* byte_offset);

bool writeProtoMsgToStream(const google::protobuf::Message& message,
                           std::fstream* stream_out);

}  // namespace utils
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_PROTOBUF_UTILS_H_
```

Includes

- fstream
- glog/logging.h
- google/protobuf/message.h
- google/protobuf/message_lite.h

Included By

- *File layer_inl.h*
- *File layer_io_inl.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::utils*

File ptcloud_vis.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/ptcloud_vis.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/ptcloud_vis.h)

Program Listing for File ptcloud_vis.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/ptcloud_vis.h)

```
#ifndef VOXBLOX_ROS_PTCLOUD_VIS_H_
#define VOXBLOX_ROS_PTCLOUD_VIS_H_

#include <algorithm>
#include <string>

#include <eigen_conversions/eigen_msg.h>
#include <pcl/point_types.h>
#include <pcl_ros/point_cloud.h>
#include <visualization_msgs/Marker.h>
#include <visualization_msgs/MarkerArray.h>
```

(continues on next page)

(continued from previous page)

```
#include <voxblox/core/common.h>
#include <voxblox/core/layer.h>
#include <voxblox/core/voxel.h>

#include "voxblox_ros/conversions.h"

namespace voxblox {

namespace ph = std::placeholders;

template <typename VoxelType>
using ShouldVisualizeVoxelColorFunctionType = std::function<bool(
    const VoxelType& voxel, const Point& coord, Color* color)>;

template <typename VoxelType>
using ShouldVisualizeVoxelIntensityFunctionType = std::function<bool(
    const VoxelType& voxel, const Point& coord, double* intensity)>;

template <typename VoxelType>
using ShouldVisualizeVoxelFunctionType =
    std::function<bool(const VoxelType& voxel, const Point& coord)>;  // NOLINT;

template <typename VoxelType>
void createColorPointcloudFromLayer(
    const Layer<VoxelType>& layer,
    const ShouldVisualizeVoxelColorFunctionType<VoxelType>& vis_function,
    pcl::PointCloud<pcl::PointXYZRGB>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  pointcloud->clear();
  BlockIndexList blocks;
  layer.getAllAllocatedBlocks(&blocks);

  // Cache layer settings.
  size_t vps = layer.voxels_per_side();
  size_t num_voxels_per_block = vps * vps * vps;

  // Temp variables.
  Color color;
  // Iterate over all blocks.
  for (const BlockIndex& index : blocks) {
    // Iterate over all voxels in said blocks.
    const Block<VoxelType>& block = layer.getBlockByIndex(index);

    for (size_t linear_index = 0; linear_index < num_voxels_per_block;
         ++linear_index) {
      Point coord = block.computeCoordinatesFromLinearIndex(linear_index);
      if (vis_function(block.getVoxelByLinearIndex(linear_index), coord,
                       &color)) {
        pcl::PointXYZRGB point;
        point.x = coord.x();
        point.y = coord.y();
        point.z = coord.z();
        point.r = color.r;
        point.g = color.g;
        point.b = color.b;
        pointcloud->push_back(point);
      }
```

(continues on next page)

(continued from previous page)

```
    }
  }
}

template <typename VoxelType>
void createColorPointcloudFromLayer(
    const Layer<VoxelType>& layer,
    const ShouldVisualizeVoxelIntensityFunctionType<VoxelType>& vis_function,
    pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  pointcloud->clear();
  BlockIndexList blocks;
  layer.getAllAllocatedBlocks(&blocks);

  // Cache layer settings.
  size_t vps = layer.voxels_per_side();
  size_t num_voxels_per_block = vps * vps * vps;

  // Temp variables.
  double intensity = 0.0;
  // Iterate over all blocks.
  for (const BlockIndex& index : blocks) {
    // Iterate over all voxels in said blocks.
    const Block<VoxelType>& block = layer.getBlockByIndex(index);

    for (size_t linear_index = 0; linear_index < num_voxels_per_block;
         ++linear_index) {
      Point coord = block.computeCoordinatesFromLinearIndex(linear_index);
      if (vis_function(block.getVoxelByLinearIndex(linear_index), coord,
                       &intensity)) {
        pcl::PointXYZI point;
        point.x = coord.x();
        point.y = coord.y();
        point.z = coord.z();
        point.intensity = intensity;
        pointcloud->push_back(point);
      }
    }
  }
}

template <typename VoxelType>
void createOccupancyBlocksFromLayer(
    const Layer<VoxelType>& layer,
    const ShouldVisualizeVoxelFunctionType<VoxelType>& vis_function,
    const std::string& frame_id,
    visualization_msgs::MarkerArray* marker_array) {
  CHECK_NOTNULL(marker_array);
  // Cache layer settings.
  size_t vps = layer.voxels_per_side();
  size_t num_voxels_per_block = vps * vps * vps;
  FloatingPoint voxel_size = layer.voxel_size();

  visualization_msgs::Marker block_marker;
  block_marker.header.frame_id = frame_id;
  block_marker.ns = "occupied_voxels";
  block_marker.id = 0;
```

(continues on next page)

(continued from previous page)

```
  block_marker.type = visualization_msgs::Marker::CUBE_LIST;
  block_marker.scale.x = block_marker.scale.y = block_marker.scale.z =
      voxel_size;
  block_marker.action = visualization_msgs::Marker::ADD;

  BlockIndexList blocks;
  layer.getAllAllocatedBlocks(&blocks);
  for (const BlockIndex& index : blocks) {
    // Iterate over all voxels in said blocks.
    const Block<VoxelType>& block = layer.getBlockByIndex(index);

    for (size_t linear_index = 0; linear_index < num_voxels_per_block;
         ++linear_index) {
      Point coord = block.computeCoordinatesFromLinearIndex(linear_index);
      if (vis_function(block.getVoxelByLinearIndex(linear_index), coord)) {
        geometry_msgs::Point cube_center;
        cube_center.x = coord.x();
        cube_center.y = coord.y();
        cube_center.z = coord.z();
        block_marker.points.push_back(cube_center);
        std_msgs::ColorRGBA color_msg;
        colorVoxbloxToMsg(rainbowColorMap((coord.z() + 2.5) / 5.0), &color_msg);
        block_marker.colors.push_back(color_msg);
      }
    }
  }
  marker_array->markers.push_back(block_marker);
}

// /Short-hand functions for visualizing different types of voxels.
inline bool visualizeNearSurfaceTsdfVoxels(const TsdfVoxel& voxel,
                                           const Point& /*coord*/,
                                           double surface_distance,
                                           Color* color) {
  CHECK_NOTNULL(color);
  constexpr float kMinWeight = 0;
  if (voxel.weight > kMinWeight &&
      std::abs(voxel.distance) < surface_distance) {
    *color = voxel.color;
    return true;
  }
  return false;
}

inline bool visualizeTsdfVoxels(const TsdfVoxel& voxel, const Point& /*coord*/,
                                Color* color) {
  CHECK_NOTNULL(color);
  constexpr float kMinWeight = 0;
  if (voxel.weight > kMinWeight) {
    *color = voxel.color;
    return true;
  }
  return false;
}

inline bool visualizeDistanceIntensityTsdfVoxels(const TsdfVoxel& voxel,
                                                 const Point& /*coord*/,
```

(continues on next page)

(continued from previous page)

```
                                                double* intensity) {
  CHECK_NOTNULL(intensity);
  constexpr float kMinWeight = 1e-3;
  if (voxel.weight > kMinWeight) {
    *intensity = voxel.distance;
    return true;
  }
  return false;
}

inline bool visualizeDistanceIntensityTsdfVoxelsNearSurface(
    const TsdfVoxel& voxel, const Point& /*coord*/, double surface_distance,
    double* intensity) {
  CHECK_NOTNULL(intensity);
  constexpr float kMinWeight = 1e-3;
  if (voxel.weight > kMinWeight &&
      std::abs(voxel.distance) < surface_distance) {
    *intensity = voxel.distance;
    return true;
  }
  return false;
}

inline bool visualizeDistanceIntensityTsdfVoxelsSlice(
    const TsdfVoxel& voxel, const Point& coord, unsigned int free_plane_index,
    FloatingPoint free_plane_val, FloatingPoint voxel_size, double* intensity) {
  CHECK_NOTNULL(intensity);
  if (std::abs(coord(free_plane_index) - free_plane_val) <=
      (voxel_size / 2.0 + kFloatEpsilon)) {
    if (voxel.weight > 1e-3) {
      *intensity = voxel.distance;
      return true;
    }
  }
  return false;
}

inline bool visualizeDistanceIntensityEsdfVoxels(const EsdfVoxel& voxel,
                                                 const Point& /*coord*/,
                                                 double* intensity) {
  CHECK_NOTNULL(intensity);
  if (voxel.observed) {
    *intensity = voxel.distance;
    return true;
  }
  return false;
}

inline bool visualizeIntensityVoxels(const IntensityVoxel& voxel,
                                     const Point& /*coord*/,
                                     double* intensity) {
  CHECK_NOTNULL(intensity);
  if (voxel.weight > 0.0) {
    *intensity = voxel.intensity;
    return true;
  }
  return false;
```

(continues on next page)

(continued from previous page)

```
}

inline bool visualizeDistanceIntensityEsdfVoxelsSlice(
    const EsdfVoxel& voxel, const Point& coord, unsigned int free_plane_index,
    FloatingPoint free_plane_val, FloatingPoint voxel_size, double* intensity) {
  CHECK_NOTNULL(intensity);

  if (std::abs(coord(free_plane_index) - free_plane_val) <=
      (voxel_size / 2.0 + kFloatEpsilon)) {
    if (voxel.observed) {
      *intensity = voxel.distance;
      return true;
    }
  }
  return false;
}

inline bool visualizeOccupiedTsdfVoxels(const TsdfVoxel& voxel,
                                        const Point& /*coord*/) {
  if (voxel.weight > 1e-3 && voxel.distance <= 0) {
    return true;
  }
  return false;
}

inline bool visualizeFreeEsdfVoxels(const EsdfVoxel& voxel,
                                    const Point& /*coord*/, float min_distance,
                                    double* intensity) {
  if (voxel.observed && voxel.distance >= min_distance) {
    *intensity = voxel.distance;
    return true;
  }
  return false;
}

inline bool visualizeOccupiedOccupancyVoxels(const OccupancyVoxel& voxel,
                                             const Point& /*coord*/) {
  const float kThresholdLogOccupancy = logOddsFromProbability(0.7);
  if (voxel.probability_log > kThresholdLogOccupancy) {
    return true;
  }
  return false;
}

// All functions that can be used directly for TSDF voxels.

inline void createSurfacePointcloudFromTsdfLayer(
    const Layer<TsdfVoxel>& layer, double surface_distance,
    pcl::PointCloud<pcl::PointXYZRGB>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<TsdfVoxel>(
      layer,
      std::bind(&visualizeNearSurfaceTsdfVoxels, ph::_1, ph::_2,
                surface_distance, ph::_3),
      pointcloud);
}
```

(continues on next page)

(continued from previous page)

```
inline void createPointcloudFromTsdfLayer(
    const Layer<TsdfVoxel>& layer,
    pcl::PointCloud<pcl::PointXYZRGB>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<TsdfVoxel>(layer, &visualizeTsdfVoxels,
                                            pointcloud);
}

inline void createDistancePointcloudFromTsdfLayer(
    const Layer<TsdfVoxel>& layer,
    pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<TsdfVoxel>(
      layer, &visualizeDistanceIntensityTsdfVoxels, pointcloud);
}

inline void createSurfaceDistancePointcloudFromTsdfLayer(
    const Layer<TsdfVoxel>& layer, double surface_distance,
    pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<TsdfVoxel>(
      layer,
      std::bind(&visualizeDistanceIntensityTsdfVoxelsNearSurface, ph::_1,
                ph::_2, surface_distance, ph::_3),
      pointcloud);
}

inline void createDistancePointcloudFromEsdfLayer(
    const Layer<EsdfVoxel>& layer,
    pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<EsdfVoxel>(
      layer, &visualizeDistanceIntensityEsdfVoxels, pointcloud);
}

inline void createFreePointcloudFromEsdfLayer(
    const Layer<EsdfVoxel>& layer, float min_distance,
    pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<EsdfVoxel>(
      layer,
      std::bind(&visualizeFreeEsdfVoxels, ph::_1, ph::_2, min_distance, ph::_3),
      pointcloud);
}

inline void createIntensityPointcloudFromIntensityLayer(
    const Layer<IntensityVoxel>& layer,
    pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  createColorPointcloudFromLayer<IntensityVoxel>(
      layer, &visualizeIntensityVoxels, pointcloud);
}

inline void createDistancePointcloudFromTsdfLayerSlice(
    const Layer<TsdfVoxel>& layer, unsigned int free_plane_index,
    FloatingPoint free_plane_val, pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
```

(continues on next page)

(continued from previous page)

```
  // Make sure that the free_plane_val doesn't fall evenly between 2 slices.
  // Prefer to push it up.
  // Using std::remainder rather than std::fmod as the latter has huge crippling
  // issues with floating-point division.
  if (std::remainder(free_plane_val, layer.voxel_size()) < kFloatEpsilon) {
    free_plane_val += layer.voxel_size() / 2.0;
  }

  createColorPointcloudFromLayer<TsdfVoxel>(
      layer,
      std::bind(&visualizeDistanceIntensityTsdfVoxelsSlice, ph::_1, ph::_2,
                free_plane_index, free_plane_val, layer.voxel_size(), ph::_3),
      pointcloud);
}

inline void createDistancePointcloudFromEsdfLayerSlice(
    const Layer<EsdfVoxel>& layer, unsigned int free_plane_index,
    FloatingPoint free_plane_val, pcl::PointCloud<pcl::PointXYZI>* pointcloud) {
  CHECK_NOTNULL(pointcloud);
  // Make sure that the free_plane_val doesn't fall evenly between 2 slices.
  // Prefer to push it up.
  // Using std::remainder rather than std::fmod as the latter has huge crippling
  // issues with floating-point division.
  if (std::remainder(free_plane_val, layer.voxel_size()) < kFloatEpsilon) {
    free_plane_val += layer.voxel_size() / 2.0;
  }

  createColorPointcloudFromLayer<EsdfVoxel>(
      layer,
      std::bind(&visualizeDistanceIntensityEsdfVoxelsSlice, ph::_1, ph::_2,
                free_plane_index, free_plane_val, layer.voxel_size(), ph::_3),
      pointcloud);
}

inline void createOccupancyBlocksFromTsdfLayer(
    const Layer<TsdfVoxel>& layer, const std::string& frame_id,
    visualization_msgs::MarkerArray* marker_array) {
  CHECK_NOTNULL(marker_array);
  createOccupancyBlocksFromLayer<TsdfVoxel>(layer, &visualizeOccupiedTsdfVoxels,
                                            frame_id, marker_array);
}

inline void createOccupancyBlocksFromOccupancyLayer(
    const Layer<OccupancyVoxel>& layer, const std::string& frame_id,
    visualization_msgs::MarkerArray* marker_array) {
  CHECK_NOTNULL(marker_array);
  createOccupancyBlocksFromLayer<OccupancyVoxel>(
      layer, &visualizeOccupiedOccupancyVoxels, frame_id, marker_array);
}

}  // namespace voxblox

#endif  // VOXBLOX_ROS_PTCLOUD_VIS_H_
```

Includes

- algorithm
- eigen_conversions/eigen_msg.h
- pcl/point_types.h
- pcl_ros/point_cloud.h
- string
- visualization_msgs/Marker.h
- visualization_msgs/MarkerArray.h
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox_ros/conversions.h (*File conversions.h*)

Included By

- *File simulation_server.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

File README.md

Contents

- *Definition (voxblox/include/voxblox/mesh/README.md)*

Definition (voxblox/include/voxblox/mesh/README.md)

Program Listing for File README.md

Return to documentation for file (voxblox/include/voxblox/mesh/README.md)

```
# NOTICE

This code is mostly adapted from OpenChisel:
   https://github.com/personalrobotics/OpenChisel
We've retained the license headers whenever relevant.
```

File ros_params.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/ros_params.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox_ros/include/voxblox_ros/ros_params.h)

Program Listing for File ros_params.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/ros_params.h)

```
#ifndef VOXBLOX_ROS_ROS_PARAMS_H_
#define VOXBLOX_ROS_ROS_PARAMS_H_

#include <ros/node_handle.h>

#include <voxblox/alignment/icp.h>
#include <voxblox/core/esdf_map.h>
#include <voxblox/core/tsdf_map.h>
#include <voxblox/integrator/esdf_integrator.h>
#include <voxblox/integrator/tsdf_integrator.h>

namespace voxblox {

inline TsdfMap::Config getTsdfMapConfigFromRosParam(
    const ros::NodeHandle& nh_private) {
  TsdfMap::Config tsdf_config;

  double voxel_size = tsdf_config.tsdf_voxel_size;
  int voxels_per_side = tsdf_config.tsdf_voxels_per_side;
  nh_private.param("tsdf_voxel_size", voxel_size, voxel_size);
  nh_private.param("tsdf_voxels_per_side", voxels_per_side, voxels_per_side);
  if (!isPowerOfTwo(voxels_per_side)) {
    ROS_ERROR("voxels_per_side must be a power of 2, setting to default value");
    voxels_per_side = tsdf_config.tsdf_voxels_per_side;
  }

  tsdf_config.tsdf_voxel_size = static_cast<FloatingPoint>(voxel_size);
  tsdf_config.tsdf_voxels_per_side = voxels_per_side;

  return tsdf_config;
}

inline ICP::Config getICPConfigFromRosParam(const ros::NodeHandle& nh_private) {
  ICP::Config icp_config;

  nh_private.param("icp_min_match_ratio", icp_config.min_match_ratio,
                   icp_config.min_match_ratio);
```

(continues on next page)

(continued from previous page)

```
  nh_private.param("icp_subsample_keep_ratio", icp_config.subsample_keep_ratio,
                   icp_config.subsample_keep_ratio);
  nh_private.param("icp_mini_batch_size", icp_config.mini_batch_size,
                   icp_config.mini_batch_size);
  nh_private.param("icp_refine_roll_pitch", icp_config.refine_roll_pitch,
                   icp_config.refine_roll_pitch);
  nh_private.param("icp_inital_translation_weighting",
                   icp_config.inital_translation_weighting,
                   icp_config.inital_translation_weighting);
  nh_private.param("icp_inital_rotation_weighting",
                   icp_config.inital_rotation_weighting,
                   icp_config.inital_rotation_weighting);

  return icp_config;
}

inline TsdfIntegratorBase::Config getTsdfIntegratorConfigFromRosParam(
    const ros::NodeHandle& nh_private) {
  TsdfIntegratorBase::Config integrator_config;

  integrator_config.voxel_carving_enabled = true;

  const TsdfMap::Config tsdf_config = getTsdfMapConfigFromRosParam(nh_private);
  integrator_config.default_truncation_distance =
      tsdf_config.tsdf_voxel_size * 4;

  double truncation_distance = integrator_config.default_truncation_distance;
  double max_weight = integrator_config.max_weight;
  nh_private.param("voxel_carving_enabled",
                   integrator_config.voxel_carving_enabled,
                   integrator_config.voxel_carving_enabled);
  nh_private.param("truncation_distance", truncation_distance,
                   truncation_distance);
  nh_private.param("max_ray_length_m", integrator_config.max_ray_length_m,
                   integrator_config.max_ray_length_m);
  nh_private.param("min_ray_length_m", integrator_config.min_ray_length_m,
                   integrator_config.min_ray_length_m);
  nh_private.param("max_weight", max_weight, max_weight);
  nh_private.param("use_const_weight", integrator_config.use_const_weight,
                   integrator_config.use_const_weight);
  nh_private.param("allow_clear", integrator_config.allow_clear,
                   integrator_config.allow_clear);
  nh_private.param("start_voxel_subsampling_factor",
                   integrator_config.start_voxel_subsampling_factor,
                   integrator_config.start_voxel_subsampling_factor);
  nh_private.param("max_consecutive_ray_collisions",
                   integrator_config.max_consecutive_ray_collisions,
                   integrator_config.max_consecutive_ray_collisions);
  nh_private.param("clear_checks_every_n_frames",
                   integrator_config.clear_checks_every_n_frames,
                   integrator_config.clear_checks_every_n_frames);
  nh_private.param("max_integration_time_s",
                   integrator_config.max_integration_time_s,
                   integrator_config.max_integration_time_s);
  nh_private.param("anti_grazing", integrator_config.enable_anti_grazing,
                   integrator_config.enable_anti_grazing);
  integrator_config.default_truncation_distance =
```

(continues on next page)

(continued from previous page)

```
      static_cast<float>(truncation_distance);
  integrator_config.max_weight = static_cast<float>(max_weight);

  return integrator_config;
}

inline EsdfMap::Config getEsdfMapConfigFromRosParam(
    const ros::NodeHandle& nh_private) {
  EsdfMap::Config esdf_config;

  const TsdfMap::Config tsdf_config = getTsdfMapConfigFromRosParam(nh_private);
  esdf_config.esdf_voxel_size = tsdf_config.tsdf_voxel_size;
  esdf_config.esdf_voxels_per_side = tsdf_config.tsdf_voxels_per_side;

  return esdf_config;
}

inline EsdfIntegrator::Config getEsdfIntegratorConfigFromRosParam(
    const ros::NodeHandle& nh_private) {
  EsdfIntegrator::Config esdf_integrator_config;

  TsdfIntegratorBase::Config tsdf_integrator_config =
      getTsdfIntegratorConfigFromRosParam(nh_private);

  esdf_integrator_config.min_distance_m =
      tsdf_integrator_config.default_truncation_distance / 2.0;

  nh_private.param("esdf_euclidean_distance",
                   esdf_integrator_config.full_euclidean_distance,
                   esdf_integrator_config.full_euclidean_distance);
  nh_private.param("esdf_max_distance_m", esdf_integrator_config.max_distance_m,
                   esdf_integrator_config.max_distance_m);
  nh_private.param("esdf_min_distance_m", esdf_integrator_config.min_distance_m,
                   esdf_integrator_config.min_distance_m);
  nh_private.param("esdf_default_distance_m",
                   esdf_integrator_config.default_distance_m,
                   esdf_integrator_config.default_distance_m);
  nh_private.param("esdf_min_diff_m", esdf_integrator_config.min_diff_m,
                   esdf_integrator_config.min_diff_m);
  nh_private.param("clear_sphere_radius",
                   esdf_integrator_config.clear_sphere_radius,
                   esdf_integrator_config.clear_sphere_radius);
  nh_private.param("occupied_sphere_radius",
                   esdf_integrator_config.occupied_sphere_radius,
                   esdf_integrator_config.occupied_sphere_radius);
  nh_private.param("esdf_add_occupied_crust",
                   esdf_integrator_config.add_occupied_crust,
                   esdf_integrator_config.add_occupied_crust);
  if (esdf_integrator_config.default_distance_m <
      esdf_integrator_config.max_distance_m) {
    esdf_integrator_config.default_distance_m =
        esdf_integrator_config.max_distance_m;
  }

  return esdf_integrator_config;
}
```

(continues on next page)

(continued from previous page)

```
}  // namespace voxblox

#endif  // VOXBLOX_ROS_ROS_PARAMS_H_
```

Includes

- ros/node_handle.h
- voxblox/alignment/icp.h (*File icp.h*)
- voxblox/core/esdf_map.h (*File esdf_map.h*)
- voxblox/core/tsdf_map.h (*File tsdf_map.h*)
- voxblox/integrator/esdf_integrator.h (*File esdf_integrator.h*)
- voxblox/integrator/tsdf_integrator.h (*File tsdf_integrator.h*)

Included By

- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

File sdf_ply.h

Contents

- *Definition (voxblox/include/voxblox/io/sdf_ply.h)*
- *Includes*
- *Namespaces*

Definition (voxblox/include/voxblox/io/sdf_ply.h)

Program Listing for File sdf_ply.h

Return to documentation for file (voxblox/include/voxblox/io/sdf_ply.h)

```
#ifndef VOXBLOX_IO_SDF_PLY_H_
#define VOXBLOX_IO_SDF_PLY_H_

#include <algorithm>
#include <string>

#include "voxblox/core/layer.h"
```

(continues on next page)

(continued from previous page)

```
#include "voxblox/io/mesh_ply.h"
#include "voxblox/mesh/mesh.h"
#include "voxblox/mesh/mesh_integrator.h"
#include "voxblox/mesh/mesh_layer.h"

namespace voxblox {

namespace io {

enum PlyOutputTypes {
  // The full SDF colorized by the distance in each voxel.
  kSdfColoredDistanceField,
  // Output isosurface, i.e. the mesh for sdf voxel types.
  kSdfIsosurface,
  // Output isosurface, i.e. the mesh for sdf voxel types.
  // Close vertices are connected and zero surface faces are removed.
  kSdfIsosurfaceConnected
};

template <typename VoxelType>
bool getColorFromVoxel(const VoxelType& voxel, const float sdf_color_range,
                       const float sdf_max_value, Color* color);

template <>
bool getColorFromVoxel(const TsdfVoxel& voxel, const float sdf_color_range,
                       const float sdf_max_value, Color* color);

template <>
bool getColorFromVoxel(const EsdfVoxel& voxel, const float sdf_color_range,
                       const float sdf_max_value, Color* color);

template <typename VoxelType>
bool convertVoxelGridToPointCloud(const Layer<VoxelType>& layer,
                                  const float sdf_color_range,
                                  const float sdf_max_value,
                                  voxblox::Mesh* point_cloud) {
  CHECK_NOTNULL(point_cloud);
  CHECK_GT(sdf_color_range, 0.0f);

  BlockIndexList blocks;
  layer.getAllAllocatedBlocks(&blocks);

  // Iterate over all blocks.
  for (const BlockIndex& index : blocks) {
    // Iterate over all voxels in said blocks.
    const Block<VoxelType>& block = layer.getBlockByIndex(index);

    const int vps = block.voxels_per_side();

    VoxelIndex voxel_index = VoxelIndex::Zero();
    for (voxel_index.x() = 0; voxel_index.x() < vps; ++voxel_index.x()) {
      for (voxel_index.y() = 0; voxel_index.y() < vps; ++voxel_index.y()) {
        for (voxel_index.z() = 0; voxel_index.z() < vps; ++voxel_index.z()) {
          const VoxelType& voxel = block.getVoxelByVoxelIndex(voxel_index);

          // Get back the original coordinate of this voxel.
          const Point coord =
```

(continues on next page)

(continued from previous page)

```
              block.computeCoordinatesFromVoxelIndex(voxel_index);

          Color color;
          if (getColorFromVoxel(voxel, sdf_color_range, sdf_max_value,
                                &color)) {
            point_cloud->vertices.push_back(coord);
            point_cloud->colors.push_back(color);
          }
        }
      }
    }
  }
  return point_cloud->size() > 0u;
}

template <typename VoxelType>
bool convertVoxelGridToPointCloud(const Layer<VoxelType>& layer,
                                  const float sdf_color_range,
                                  voxblox::Mesh* point_cloud) {
  constexpr float kInvalidSdfMaxValue = -1.0f;
  return convertVoxelGridToPointCloud<VoxelType>(
      layer, sdf_color_range, kInvalidSdfMaxValue, point_cloud);
}

template <typename VoxelType>
bool convertLayerToMesh(
    const Layer<VoxelType>& layer, const MeshIntegratorConfig& mesh_config,
    voxblox::Mesh* mesh, const bool connected_mesh = true,
    const FloatingPoint vertex_proximity_threshold = 1e-10) {
  CHECK_NOTNULL(mesh);

  MeshLayer mesh_layer(layer.block_size());
  MeshIntegrator<VoxelType> mesh_integrator(mesh_config, layer, &mesh_layer);

  // Generate mesh layer.
  constexpr bool only_mesh_updated_blocks = false;
  constexpr bool clear_updated_flag = false;
  mesh_integrator.generateMesh(only_mesh_updated_blocks, clear_updated_flag);

  // Extract mesh from mesh layer, either by simply concatenating all meshes
  // (there is one per block) or by connecting them.
  if (connected_mesh) {
    mesh_layer.getConnectedMesh(mesh, vertex_proximity_threshold);
  } else {
    mesh_layer.getMesh(mesh);
  }
  return mesh->size() > 0u;
}
template <typename VoxelType>
bool convertLayerToMesh(
    const Layer<VoxelType>& layer, voxblox::Mesh* mesh,
    const bool connected_mesh = true,
    const FloatingPoint vertex_proximity_threshold = 1e-10) {
  MeshIntegratorConfig mesh_config;
  return convertLayerToMesh(layer, mesh_config, mesh, connected_mesh,
                            vertex_proximity_threshold);
}
```

(continues on next page)

(continued from previous page)

```
template <typename VoxelType>
bool outputLayerAsPly(const Layer<VoxelType>& layer,
                      const std::string& filename, PlyOutputTypes type,
                      const float sdf_color_range = 0.3f,
                      const float max_sdf_value_to_output = 0.3f) {
  CHECK(!filename.empty());
  switch (type) {
    case PlyOutputTypes::kSdfColoredDistanceField: {
      voxblox::Mesh point_cloud;
      if (!convertVoxelGridToPointCloud(
              layer, sdf_color_range, max_sdf_value_to_output, &point_cloud)) {
        return false;
      }

      return outputMeshAsPly(filename, point_cloud);
    }
    case PlyOutputTypes::kSdfIsosurface: {
      constexpr bool kConnectedMesh = false;

      voxblox::Mesh mesh;
      if (!convertLayerToMesh(layer, &mesh, kConnectedMesh)) {
        return false;
      }
      return outputMeshAsPly(filename, mesh);
    }
    case PlyOutputTypes::kSdfIsosurfaceConnected: {
      constexpr bool kConnectedMesh = true;

      voxblox::Mesh mesh;
      if (!convertLayerToMesh(layer, &mesh, kConnectedMesh)) {
        return false;
      }
      return outputMeshAsPly(filename, mesh);
    }

    default:
      LOG(FATAL) << "Unknown layer to ply output type: "
                 << static_cast<int>(type);
  }
  return false;
}

}  // namespace io

}  // namespace voxblox

#endif  // VOXBLOX_IO_SDF_PLY_H_
```

Includes

- algorithm
- string
- voxblox/core/layer.h (*File layer.h*)

- `voxblox/io/mesh_ply.h` (*File mesh_ply.h*)
- `voxblox/mesh/mesh.h` (*File mesh.h*)
- `voxblox/mesh/mesh_integrator.h` (*File mesh_integrator.h*)
- `voxblox/mesh/mesh_layer.h` (*File mesh_layer.h*)

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::io*

File simulation_server.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/simulation_server.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox_ros/include/voxblox_ros/simulation_server.h)

Program Listing for File simulation_server.h

Return to documentation for file (`voxblox_ros/include/voxblox_ros/simulation_server.h`)

```
#ifndef VOXBLOX_ROS_SIMULATION_SERVER_H_
#define VOXBLOX_ROS_SIMULATION_SERVER_H_

#include <ros/ros.h>
#include <memory>
#include <string>

#include <voxblox/core/esdf_map.h>
#include <voxblox/core/tsdf_map.h>
#include <voxblox/integrator/esdf_integrator.h>
#include <voxblox/integrator/esdf_occ_integrator.h>
#include <voxblox/integrator/occupancy_integrator.h>
#include <voxblox/integrator/tsdf_integrator.h>
#include <voxblox/mesh/mesh_integrator.h>
#include <voxblox/simulation/simulation_world.h>

#include "voxblox_ros/conversions.h"
#include "voxblox_ros/mesh_vis.h"
#include "voxblox_ros/ptcloud_vis.h"
#include "voxblox_ros/ros_params.h"

namespace voxblox {
```

(continues on next page)

(continued from previous page)

```
class SimulationServer {
 public:
  SimulationServer(const ros::NodeHandle& nh,
                   const ros::NodeHandle& nh_private);

  SimulationServer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private,
                   const EsdfMap::Config& esdf_config,
                   const EsdfIntegrator::Config& esdf_integrator_config,
                   const TsdfMap::Config& tsdf_config,
                   const TsdfIntegratorBase::Config& tsdf_integrator_config);

  virtual ~SimulationServer() {}

  void run();

  virtual void prepareWorld() = 0;

  void generateSDF();

  void evaluate();

  void visualize();

 protected:
  void getServerConfigFromRosParam(const ros::NodeHandle& nh_private);

  bool generatePlausibleViewpoint(FloatingPoint min_distance, Point* ray_origin,
                                  Point* ray_direction) const;

  ros::NodeHandle nh_;
  ros::NodeHandle nh_private_;

  // A bunch of publishers :)
  ros::Publisher sim_pub_;
  ros::Publisher tsdf_gt_pub_;
  ros::Publisher esdf_gt_pub_;
  ros::Publisher tsdf_gt_mesh_pub_;
  ros::Publisher tsdf_test_pub_;
  ros::Publisher esdf_test_pub_;
  ros::Publisher tsdf_test_mesh_pub_;
  ros::Publisher view_ptcloud_pub_;

  // Settings
  FloatingPoint voxel_size_;
  int voxels_per_side_;
  std::string world_frame_;
  bool generate_occupancy_;
  bool visualize_;
  FloatingPoint visualization_slice_level_;
  bool generate_mesh_;
  bool incremental_;
  bool add_robot_pose_;
  FloatingPoint truncation_distance_;
  FloatingPoint esdf_max_distance_;
  size_t max_attempts_to_generate_viewpoint_;
```

(continues on next page)

(continued from previous page)

```
  // Camera settings
  Eigen::Vector2i depth_camera_resolution_;
  FloatingPoint fov_h_rad_;
  FloatingPoint max_dist_;
  FloatingPoint min_dist_;
  int num_viewpoints_;

  // Actual simulation server.
  SimulationWorld world_;

  // Maps (GT and generates from sensors) generated here.
  std::unique_ptr<Layer<TsdfVoxel> > tsdf_gt_;
  std::unique_ptr<Layer<EsdfVoxel> > esdf_gt_;

  // Generated maps:
  std::unique_ptr<Layer<TsdfVoxel> > tsdf_test_;
  std::unique_ptr<Layer<EsdfVoxel> > esdf_test_;
  std::unique_ptr<Layer<OccupancyVoxel> > occ_test_;

  // Integrators:
  std::unique_ptr<TsdfIntegratorBase> tsdf_integrator_;
  std::unique_ptr<EsdfIntegrator> esdf_integrator_;
  std::unique_ptr<OccupancyIntegrator> occ_integrator_;
  std::unique_ptr<EsdfOccIntegrator> esdf_occ_integrator_;
};

}  // namespace voxblox

#endif  // VOXBLOX_ROS_SIMULATION_SERVER_H_
```

Includes

- memory
- ros/ros.h
- string
- voxblox/core/esdf_map.h (*File esdf_map.h*)
- voxblox/core/tsdf_map.h (*File tsdf_map.h*)
- voxblox/integrator/esdf_integrator.h (*File esdf_integrator.h*)
- voxblox/integrator/esdf_occ_integrator.h (*File esdf_occ_integrator.h*)
- voxblox/integrator/occupancy_integrator.h (*File occupancy_integrator.h*)
- voxblox/integrator/tsdf_integrator.h (*File tsdf_integrator.h*)
- voxblox/mesh/mesh_integrator.h (*File mesh_integrator.h*)
- voxblox/simulation/simulation_world.h (*File simulation_world.h*)
- voxblox_ros/conversions.h (*File conversions.h*)
- voxblox_ros/mesh_vis.h (*File mesh_vis.h*)
- voxblox_ros/ptcloud_vis.h (*File ptcloud_vis.h*)
- voxblox_ros/ros_params.h (*File ros_params.h*)

Namespaces

- *Namespace voxblox*

Classes

- *Class SimulationServer*

File simulation_world.h

Contents

- *Definition (voxblox/include/voxblox/simulation/simulation_world.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/simulation/simulation_world.h)

Program Listing for File simulation_world.h

Return to documentation for file (voxblox/include/voxblox/simulation/simulation_world.h)

```
#ifndef VOXBLOX_SIMULATION_SIMULATION_WORLD_H_
#define VOXBLOX_SIMULATION_SIMULATION_WORLD_H_

#include <list>
#include <memory>
#include <random>
#include <vector>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/simulation/objects.h"

namespace voxblox {

class SimulationWorld {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  SimulationWorld();
  virtual ~SimulationWorld() {}

  void addObject(std::unique_ptr<Object> object);
```

(continues on next page)

(continued from previous page)

```
  void addGroundLevel(FloatingPoint height);

  void addPlaneBoundaries(FloatingPoint x_min, FloatingPoint x_max,
                          FloatingPoint y_min, FloatingPoint y_max);

  void clear();

  void getPointcloudFromViewpoint(const Point& view_origin,
                                  const Point& view_direction,
                                  const Eigen::Vector2i& camera_res,
                                  FloatingPoint fov_h_rad,
                                  FloatingPoint max_dist, Pointcloud* ptcloud,
                                  Colors* colors) const;
  void getPointcloudFromTransform(const Transformation& pose,
                                  const Eigen::Vector2i& camera_res,
                                  FloatingPoint fov_h_rad,
                                  FloatingPoint max_dist, Pointcloud* ptcloud,
                                  Colors* colors) const;

  void getNoisyPointcloudFromViewpoint(const Point& view_origin,
                                       const Point& view_direction,
                                       const Eigen::Vector2i& camera_res,
                                       FloatingPoint fov_h_rad,
                                       FloatingPoint max_dist,
                                       FloatingPoint noise_sigma,
                                       Pointcloud* ptcloud, Colors* colors);
  void getNoisyPointcloudFromTransform(const Transformation& pose,
                                       const Eigen::Vector2i& camera_res,
                                       FloatingPoint fov_h_rad,
                                       FloatingPoint max_dist,
                                       FloatingPoint noise_sigma,
                                       Pointcloud* ptcloud, Colors* colors);

  // === Computing ground truth SDFs ===
  template <typename VoxelType>
  void generateSdfFromWorld(FloatingPoint max_dist,
                            Layer<VoxelType>* layer) const;

  FloatingPoint getDistanceToPoint(const Point& coords,
                                   FloatingPoint max_dist) const;

  void setBounds(const Point& min_bound, const Point& max_bound) {
    min_bound_ = min_bound;
    max_bound_ = max_bound;
  }

  Point getMinBound() const { return min_bound_; }
  Point getMaxBound() const { return max_bound_; }

 private:
  template <typename VoxelType>
  void setVoxel(FloatingPoint dist, const Color& color, VoxelType* voxel) const;

  FloatingPoint getNoise(FloatingPoint noise_sigma);

  std::list<std::unique_ptr<Object> > objects_;
```

(continues on next page)

(continued from previous page)

```
  // World boundaries... Can be changed arbitrarily, just sets ground truth
  // generation and visualization bounds, accurate only up to block size.
  Point min_bound_;
  Point max_bound_;

  std::default_random_engine generator_;
};

}  // namespace voxblox

#endif  // VOXBLOX_SIMULATION_SIMULATION_WORLD_H_

#include "voxblox/simulation/simulation_world_inl.h"
```

Includes

- list
- memory
- random
- vector
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/simulation/objects.h (*File objects.h*)
- voxblox/simulation/simulation_world_inl.h (*File simulation_world_inl.h*)

Included By

- *File simulation_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class SimulationWorld*

File simulation_world_inl.h

Contents

- *Definition (voxblox/include/voxblox/simulation/simulation_world_inl.h)*

- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/simulation/simulation_world_inl.h)

Program Listing for File simulation_world_inl.h

Return to documentation for file (voxblox/include/voxblox/simulation/simulation_world_inl.h)

```
#ifndef VOXBLOX_SIMULATION_SIMULATION_WORLD_INL_H_
#define VOXBLOX_SIMULATION_SIMULATION_WORLD_INL_H_

#include <algorithm>
#include <iostream>
#include <memory>

#include "voxblox/core/block.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

template <typename VoxelType>
void SimulationWorld::generateSdfFromWorld(FloatingPoint max_dist,
                                           Layer<VoxelType>* layer) const {
  timing::Timer sim_timer("sim/generate_sdf");

  CHECK_NOTNULL(layer);
  // Iterate over every voxel in the layer and compute its distance to all
  // objects.

  // Get all blocks within bounds. For now, only respect bounds approximately:
  // that is, up to block boundaries.
  FloatingPoint block_size = layer->block_size();
  FloatingPoint half_block_size = block_size / 2.0;

  BlockIndexList blocks;
  for (FloatingPoint x = min_bound_.x() - half_block_size;
       x <= max_bound_.x() + half_block_size; x += block_size) {
    for (FloatingPoint y = min_bound_.y() - half_block_size;
         y <= max_bound_.y() + half_block_size; y += block_size) {
      for (FloatingPoint z = min_bound_.z() - half_block_size;
           z <= max_bound_.z() + half_block_size; z += block_size) {
        blocks.push_back(
            layer->computeBlockIndexFromCoordinates(Point(x, y, z)));
      }
    }
  }

  // Iterate over all blocks filling this stuff in.
  for (const BlockIndex& block_index : blocks) {
    typename Block<VoxelType>::Ptr block =
```

(continues on next page)

(continued from previous page)

```
        layer->allocateBlockPtrByIndex(block_index);
    for (size_t i = 0; i < block->num_voxels(); ++i) {
      VoxelType& voxel = block->getVoxelByLinearIndex(i);
      Point coords = block->computeCoordinatesFromLinearIndex(i);
      // Check that it's in bounds, otherwise skip it.
      if (!(coords.x() >= min_bound_.x() && coords.x() <= max_bound_.x() &&
            coords.y() >= min_bound_.y() && coords.y() <= max_bound_.y() &&
            coords.z() >= min_bound_.z() && coords.z() <= max_bound_.z())) {
        continue;
      }

      // Iterate over all objects and get distances to this thing.
      FloatingPoint voxel_dist = max_dist;
      Color color;
      for (const std::unique_ptr<Object>& object : objects_) {
        FloatingPoint object_dist = object->getDistanceToPoint(coords);
        if (object_dist < voxel_dist) {
          voxel_dist = object_dist;
          color = object->getColor();
        }
      }

      // Then update the thing.
      voxel_dist = std::max(voxel_dist, -max_dist);
      setVoxel(voxel_dist, color, &voxel);
    }
  }
}

template <>
void SimulationWorld::setVoxel(FloatingPoint dist, const Color& color,
                               TsdfVoxel* voxel) const {
  voxel->distance = static_cast<float>(dist);
  voxel->color = color;
  voxel->weight = 1.0f;  // Just to make sure it gets visualized/meshed/etc.
}

// Color ignored.
template <>
void SimulationWorld::setVoxel(FloatingPoint dist, const Color& /*color*/,
                               EsdfVoxel* voxel) const {
  voxel->distance = static_cast<float>(dist);
  voxel->observed = true;
}

}  // namespace voxblox

#endif  // VOXBLOX_SIMULATION_SIMULATION_WORLD_INL_H_
```

Includes

- algorithm
- iostream
- memory

- `voxblox/core/block.h` (*File block.h*)
- `voxblox/utils/timing.h` (*File timing.h*)

Included By

- *File simulation_world.h*

Namespaces

- *Namespace voxblox*

File timing.h

Contents

- *Definition (`voxblox/include/voxblox/utils/timing.h`)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (`voxblox/include/voxblox/utils/timing.h`)

Program Listing for File timing.h

Return to documentation for file (`voxblox/include/voxblox/utils/timing.h`)

```
/*
 * Copyright (C) 2012-2013 Simon Lynen, ASL, ETH Zurich, Switzerland
 * You can contact the author at <slynen at ethz dot ch>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/* Adapted from Paul Furgale Schweizer Messer sm_timing */

#ifndef VOXBLOX_UTILS_TIMING_H_
```

(continues on next page)

(continued from previous page)

```
#define VOXBLOX_UTILS_TIMING_H_

#include <algorithm>
#include <chrono>
#include <limits>
#include <map>
#include <mutex>
#include <string>
#include <vector>

#include "voxblox/core/common.h"

namespace voxblox {

namespace timing {

template <typename T, typename Total, int N>
class Accumulator {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  Accumulator()
      : window_samples_(0),
        totalsamples_(0),
        window_sum_(0),
        sum_(0),
        min_(std::numeric_limits<T>::max()),
        max_(std::numeric_limits<T>::min()) {}

  void Add(T sample) {
    if (window_samples_ < N) {
      samples_[window_samples_++] = sample;
      window_sum_ += sample;
    } else {
      T& oldest = samples_[window_samples_++ % N];
      window_sum_ += sample - oldest;
      oldest = sample;
    }
    sum_ += sample;
    ++totalsamples_;
    if (sample > max_) {
      max_ = sample;
    }
    if (sample < min_) {
      min_ = sample;
    }
  }

  int TotalSamples() const { return totalsamples_; }

  double Sum() const { return sum_; }

  double Mean() const { return sum_ / totalsamples_; }

  double RollingMean() const {
    return window_sum_ / std::min(window_samples_, N);
  }
```

(continues on next page)

(continued from previous page)

```
  double Max() const { return max_; }

  double Min() const { return min_; }

  double LazyVariance() const {
    if (window_samples_ == 0) {
      return 0.0;
    }
    double var = 0;
    double mean = RollingMean();
    for (int i = 0; i < std::min(window_samples_, N); ++i) {
      var += (samples_[i] - mean) * (samples_[i] - mean);
    }
    var /= std::min(window_samples_, N);
    return var;
  }

 private:
  int window_samples_;
  int totalsamples_;
  Total window_sum_;
  Total sum_;
  T min_;
  T max_;
  T samples_[N];
};

struct TimerMapValue {
  TimerMapValue() {}

  Accumulator<double, double, 50> acc_;
};

class DummyTimer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  explicit DummyTimer(size_t /*handle*/, bool /*constructStopped*/ = false) {}
  explicit DummyTimer(std::string const& /*tag*/,
                      bool /*constructStopped*/ = false) {}
  ~DummyTimer() {}

  void Start() {}
  void Stop() {}
  bool IsTiming() { return false; }
};

class Timer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  explicit Timer(size_t handle, bool constructStopped = false);
  explicit Timer(std::string const& tag, bool constructStopped = false);
  ~Timer();

  void Start();
```

(continues on next page)

(continued from previous page)

```
  void Stop();
  bool IsTiming() const;

 private:
  std::chrono::time_point<std::chrono::system_clock> time_;

  bool timing_;
  size_t handle_;
};

class Timing {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  typedef std::map<std::string, size_t> map_t;
  friend class Timer;
  // Definition of static functions to query the timers.
  static size_t GetHandle(std::string const& tag);
  static std::string GetTag(size_t handle);
  static double GetTotalSeconds(size_t handle);
  static double GetTotalSeconds(std::string const& tag);
  static double GetMeanSeconds(size_t handle);
  static double GetMeanSeconds(std::string const& tag);
  static size_t GetNumSamples(size_t handle);
  static size_t GetNumSamples(std::string const& tag);
  static double GetVarianceSeconds(size_t handle);
  static double GetVarianceSeconds(std::string const& tag);
  static double GetMinSeconds(size_t handle);
  static double GetMinSeconds(std::string const& tag);
  static double GetMaxSeconds(size_t handle);
  static double GetMaxSeconds(std::string const& tag);
  static double GetHz(size_t handle);
  static double GetHz(std::string const& tag);
  static void Print(std::ostream& out);
  static std::string Print();
  static std::string SecondsToTimeString(double seconds);
  static void Reset();
  static const map_t& GetTimers() { return Instance().tagMap_; }

 private:
  void AddTime(size_t handle, double seconds);

  static Timing& Instance();

  Timing();
  ~Timing();

  typedef AlignedVector<TimerMapValue> list_t;

  list_t timers_;
  map_t tagMap_;
  size_t maxTagLength_;
  std::mutex mutex_;
};

#if ENABLE_MSF_TIMING
typedef Timer DebugTimer;
```

(continues on next page)

(continued from previous page)

```
#else
typedef DummyTimer DebugTimer;
#endif

}  // namespace timing
}  // namespace voxblox

#endif  // VOXBLOX_UTILS_TIMING_H_
```

Includes

- algorithm
- chrono
- limits
- map (*File color_maps.h*)
- mutex
- string
- vector
- voxblox/core/common.h (*File common.h*)

Included By

- *File esdf_integrator.h*
- *File esdf_occ_integrator.h*
- *File integrator_utils.h*
- *File intensity_integrator.h*
- *File occupancy_integrator.h*
- *File tsdf_integrator.h*
- *File mesh_integrator.h*
- *File simulation_world_inl.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::timing*

Classes

- *Struct TimerMapValue*
- *Template Class Accumulator*

- *Class DummyTimer*
- *Class Timer*
- *Class Timing*

File transformer.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/transformer.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox_ros/include/voxblox_ros/transformer.h)

Program Listing for File transformer.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/transformer.h)

```
#ifndef VOXBLOX_ROS_TRANSFORMER_H_
#define VOXBLOX_ROS_TRANSFORMER_H_

#include <geometry_msgs/TransformStamped.h>
#include <tf/transform_listener.h>
#include <string>

#include <voxblox/core/common.h>

namespace voxblox {

class Transformer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  Transformer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private);

  bool lookupTransform(const std::string& from_frame,
                       const std::string& to_frame, const ros::Time& timestamp,
                       Transformation* transform);

  void transformCallback(const geometry_msgs::TransformStamped& transform_msg);

 private:
  bool lookupTransformTf(const std::string& from_frame,
                         const std::string& to_frame,
                         const ros::Time& timestamp, Transformation* transform);

  bool lookupTransformQueue(const ros::Time& timestamp,
```

(continues on next page)

(continued from previous page)

```
                          Transformation* transform);

  ros::NodeHandle nh_;
  ros::NodeHandle nh_private_;

  std::string world_frame_;
  std::string sensor_frame_;
  bool use_tf_transforms_;
  int64_t timestamp_tolerance_ns_;
  Transformation T_B_C_;
  Transformation T_B_D_;
  tf::TransformListener tf_listener_;

  // 1 Only used if use_tf_transforms_ set to false.
  ros::Subscriber transform_sub_;

  // 1 Transform queue, used only when use_tf_transforms is false.
  AlignedDeque<geometry_msgs::TransformStamped> transform_queue_;
};

}  // namespace voxblox

#endif  // VOXBLOX_ROS_TRANSFORMER_H_
```

Includes

- `geometry_msgs/TransformStamped.h`
- `string`
- `tf/transform_listener.h`
- `voxblox/core/common.h` (*File common.h*)

Included By

- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class Transformer*

File tsdf_integrator.h

Contents

- *Definition (voxblox/include/voxblox/integrator/tsdf_integrator.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/integrator/tsdf_integrator.h)

Program Listing for File tsdf_integrator.h

Return to documentation for file (voxblox/include/voxblox/integrator/tsdf_integrator.h)

```
#ifndef VOXBLOX_INTEGRATOR_TSDF_INTEGRATOR_H_
#define VOXBLOX_INTEGRATOR_TSDF_INTEGRATOR_H_

#include <algorithm>
#include <atomic>
#include <cmath>
#include <deque>
#include <limits>
#include <memory>
#include <mutex>
#include <queue>
#include <string>
#include <thread>
#include <utility>
#include <vector>

#include <glog/logging.h>
#include <Eigen/Core>

#include "voxblox/core/block_hash.h"
#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"
#include "voxblox/integrator/integrator_utils.h"
#include "voxblox/utils/approx_hash_array.h"
#include "voxblox/utils/timing.h"

namespace voxblox {

enum class TsdfIntegratorType : int {
  kSimple = 1,
  kMerged = 2,
  kFast = 3,
};

static constexpr size_t kNumTsdfIntegratorTypes = 3u;

const std::array<std::string, kNumTsdfIntegratorTypes>
```

(continues on next page)

(continued from previous page)

```
    kTsdfIntegratorTypeNames = {{/*kSimple*/ "simple",
                                 /*kMerged*/ "merged",
                                 /*kFast*/ "fast"}};

class TsdfIntegratorBase {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  typedef std::shared_ptr<TsdfIntegratorBase> Ptr;

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    float default_truncation_distance = 0.1;
    float max_weight = 10000.0;
    bool voxel_carving_enabled = true;
    FloatingPoint min_ray_length_m = 0.1;
    FloatingPoint max_ray_length_m = 5.0;
    bool use_const_weight = false;
    bool allow_clear = true;
    bool use_weight_dropoff = true;
    bool use_sparsity_compensation_factor = false;
    float sparsity_compensation_factor = 1.0f;
    size_t integrator_threads = std::thread::hardware_concurrency();

    bool enable_anti_grazing = false;

    float start_voxel_subsampling_factor = 2.0f;
    int max_consecutive_ray_collisions = 2;
    int clear_checks_every_n_frames = 1;
    float max_integration_time_s = std::numeric_limits<float>::max();
  };

  TsdfIntegratorBase(const Config& config, Layer<TsdfVoxel>* layer);

  virtual void integratePointCloud(const Transformation& T_G_C,
                                   const Pointcloud& points_C,
                                   const Colors& colors,
                                   const bool freespace_points = false) = 0;

  const Config& getConfig() const { return config_; }

 protected:
  inline bool isPointValid(const Point& point_C, const bool freespace_point,
                           bool* is_clearing) const;

  TsdfVoxel* allocateStorageAndGetVoxelPtr(const GlobalIndex& global_voxel_idx,
                                           Block<TsdfVoxel>::Ptr* last_block,
                                           BlockIndex* last_block_idx);

  void updateLayerWithStoredBlocks();

  void updateTsdfVoxel(const Point& origin, const Point& point_G,
                       const GlobalIndex& global_voxel_index,
                       const Color& color, const float weight,
                       TsdfVoxel* tsdf_voxel);

  float computeDistance(const Point& origin, const Point& point_G,
```

(continues on next page)

(continued from previous page)

```
                        const Point& voxel_center) const;

  float getVoxelWeight(const Point& point_C) const;

  Config config_;

  Layer<TsdfVoxel>* layer_;

  // Cached map config.
  FloatingPoint voxel_size_;
  size_t voxels_per_side_;
  FloatingPoint block_size_;

  // Derived types.
  FloatingPoint voxel_size_inv_;
  FloatingPoint voxels_per_side_inv_;
  FloatingPoint block_size_inv_;

  std::mutex temp_block_mutex_;
  Layer<TsdfVoxel>::BlockHashMap temp_block_map_;

  ApproxHashArray<12, std::mutex, GlobalIndex, LongIndexHash> mutexes_;
};

class TsdfIntegratorFactory {
 public:
  static TsdfIntegratorBase::Ptr create(
      const std::string& integrator_type_name,
      const TsdfIntegratorBase::Config& config, Layer<TsdfVoxel>* layer);
  static TsdfIntegratorBase::Ptr create(
      const TsdfIntegratorType integrator_type,
      const TsdfIntegratorBase::Config& config, Layer<TsdfVoxel>* layer);
};

class SimpleTsdfIntegrator : public TsdfIntegratorBase {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  SimpleTsdfIntegrator(const Config& config, Layer<TsdfVoxel>* layer)
      : TsdfIntegratorBase(config, layer) {}

  void integratePointCloud(const Transformation& T_G_C,
                           const Pointcloud& points_C, const Colors& colors,
                           const bool freespace_points = false);

  void integrateFunction(const Transformation& T_G_C,
                         const Pointcloud& points_C, const Colors& colors,
                         const bool freespace_points,
                         ThreadSafeIndex* index_getter);
};

class MergedTsdfIntegrator : public TsdfIntegratorBase {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  MergedTsdfIntegrator(const Config& config, Layer<TsdfVoxel>* layer)
      : TsdfIntegratorBase(config, layer) {}
```

(continues on next page)

(continued from previous page)

```
  void integratePointCloud(const Transformation& T_G_C,
                           const Pointcloud& points_C, const Colors& colors,
                           const bool freespace_points = false);

 protected:
  void bundleRays(const Transformation& T_G_C, const Pointcloud& points_C,
                  const bool freespace_points, ThreadSafeIndex* index_getter,
                  LongIndexHashMapType<AlignedVector<size_t>>::type* voxel_map,
                  LongIndexHashMapType<AlignedVector<size_t>>::type* clear_map);

  void integrateVoxel(
      const Transformation& T_G_C, const Pointcloud& points_C,
      const Colors& colors, bool enable_anti_grazing, bool clearing_ray,
      const std::pair<GlobalIndex, AlignedVector<size_t>>& kv,
      const LongIndexHashMapType<AlignedVector<size_t>>::type& voxel_map);

  void integrateVoxels(
      const Transformation& T_G_C, const Pointcloud& points_C,
      const Colors& colors, bool enable_anti_grazing, bool clearing_ray,
      const LongIndexHashMapType<AlignedVector<size_t>>::type& voxel_map,
      const LongIndexHashMapType<AlignedVector<size_t>>::type& clear_map,
      size_t thread_idx);

  void integrateRays(
      const Transformation& T_G_C, const Pointcloud& points_C,
      const Colors& colors, bool enable_anti_grazing, bool clearing_ray,
      const LongIndexHashMapType<AlignedVector<size_t>>::type& voxel_map,
      const LongIndexHashMapType<AlignedVector<size_t>>::type& clear_map);
};

class FastTsdfIntegrator : public TsdfIntegratorBase {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  FastTsdfIntegrator(const Config& config, Layer<TsdfVoxel>* layer)
      : TsdfIntegratorBase(config, layer) {}

  void integrateFunction(const Transformation& T_G_C,
                         const Pointcloud& points_C, const Colors& colors,
                         const bool freespace_points,
                         ThreadSafeIndex* index_getter);

  void integratePointCloud(const Transformation& T_G_C,
                           const Pointcloud& points_C, const Colors& colors,
                           const bool freespace_points = false);

 private:
  static constexpr size_t masked_bits_ = 20;
  static constexpr size_t full_reset_threshold_ = 10000;

  ApproxHashSet<masked_bits_, full_reset_threshold_, GlobalIndex, LongIndexHash>
      start_voxel_approx_set_;

  ApproxHashSet<masked_bits_, full_reset_threshold_, GlobalIndex, LongIndexHash>
      voxel_observed_approx_set_;
```

(continues on next page)

(continued from previous page)

```
  std::chrono::time_point<std::chrono::steady_clock> integration_start_time_;
};

}  // namespace voxblox

#endif  // VOXBLOX_INTEGRATOR_TSDF_INTEGRATOR_H_
```

Includes

- Eigen/Core
- algorithm
- atomic
- cmath
- deque
- glog/logging.h
- limits
- memory
- mutex
- queue (*File bucket_queue.h*)
- string
- thread
- utility
- vector
- voxblox/core/block_hash.h (*File block_hash.h*)
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)
- voxblox/integrator/integrator_utils.h (*File integrator_utils.h*)
- voxblox/utils/approx_hash_array.h (*File approx_hash_array.h*)
- voxblox/utils/timing.h (*File timing.h*)

Included By

- *File mesh_vis.h*
- *File ros_params.h*
- *File simulation_server.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct TsdfIntegratorBase::Config*
- *Class FastTsdfIntegrator*
- *Class MergedTsdfIntegrator*
- *Class SimpleTsdfIntegrator*
- *Class TsdfIntegratorBase*
- *Class TsdfIntegratorFactory*

File tsdf_map.h

Contents

- *Definition (voxblox/include/voxblox/core/tsdf_map.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/core/tsdf_map.h)

Program Listing for File tsdf_map.h

Return to documentation for file (voxblox/include/voxblox/core/tsdf_map.h)

```
#ifndef VOXBLOX_CORE_TSDF_MAP_H_
#define VOXBLOX_CORE_TSDF_MAP_H_

#include <glog/logging.h>
#include <memory>
#include <string>
#include <utility>

#include "voxblox/core/common.h"
#include "voxblox/core/layer.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
class TsdfMap {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
```

(continues on next page)

(continued from previous page)

```
  typedef std::shared_ptr<TsdfMap> Ptr;

  struct Config {
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW

    FloatingPoint tsdf_voxel_size = 0.2;
    size_t tsdf_voxels_per_side = 16u;
  };

  explicit TsdfMap(const Config& config)
      : tsdf_layer_(new Layer<TsdfVoxel>(config.tsdf_voxel_size,
                                         config.tsdf_voxels_per_side)) {
    block_size_ = config.tsdf_voxel_size * config.tsdf_voxels_per_side;
  }

  explicit TsdfMap(const Layer<TsdfVoxel>& layer)
      : TsdfMap(aligned_shared<Layer<TsdfVoxel>>(layer)) {}

  explicit TsdfMap(Layer<TsdfVoxel>::Ptr layer) : tsdf_layer_(layer) {
    if (!layer) {
      /* NOTE(mereweth@jpl.nasa.gov) - throw std exception for Python to catch
       * This is idiomatic when wrapping C++ code for Python, especially with
       * pybind11
       */
      throw std::runtime_error(std::string("Null Layer<TsdfVoxel>::Ptr") +
                               " in TsdfMap constructor");
    }

    CHECK(layer);
    block_size_ = layer->block_size();
  }

  virtual ~TsdfMap() {}

  Layer<TsdfVoxel>* getTsdfLayerPtr() { return tsdf_layer_.get(); }
  const Layer<TsdfVoxel>& getTsdfLayer() const { return *tsdf_layer_; }

  FloatingPoint block_size() const { return block_size_; }
  FloatingPoint voxel_size() const { return tsdf_layer_->voxel_size(); }

  /* NOTE(mereweth@jpl.nasa.gov)
   * EigenDRef is fully dynamic stride type alias for Numpy array slices
   * Use column-major matrices; column-by-column traversal is faster
   * Convenience alias borrowed from pybind11
   */
  using EigenDStride = Eigen::Stride<Eigen::Dynamic, Eigen::Dynamic>;
  template <typename MatrixType>
  using EigenDRef = Eigen::Ref<MatrixType, 0, EigenDStride>;

  unsigned int coordPlaneSliceGetDistanceWeight(
      unsigned int free_plane_index, double free_plane_val,
      EigenDRef<Eigen::Matrix<double, 3, Eigen::Dynamic>>& positions,
      Eigen::Ref<Eigen::VectorXd> distances,
      Eigen::Ref<Eigen::VectorXd> weights, unsigned int max_points) const;

 protected:
```

(continues on next page)

(continued from previous page)

```
  FloatingPoint block_size_;

  // The layers.
  Layer<TsdfVoxel>::Ptr tsdf_layer_;
};

}  // namespace voxblox

#endif  // VOXBLOX_CORE_TSDF_MAP_H_
```

Includes

- glog/logging.h
- memory
- string
- utility
- voxblox/core/common.h (*File common.h*)
- voxblox/core/layer.h (*File layer.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Included By

- *File ros_params.h*
- *File simulation_server.h*
- *File tsdf_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Struct TsdfMap::Config*
- *Class TsdfMap*

File tsdf_server.h

Contents

- *Definition (voxblox_ros/include/voxblox_ros/tsdf_server.h)*
- *Includes*

- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox_ros/include/voxblox_ros/tsdf_server.h)

Program Listing for File tsdf_server.h

Return to documentation for file (voxblox_ros/include/voxblox_ros/tsdf_server.h)

```
#ifndef VOXBLOX_ROS_TSDF_SERVER_H_
#define VOXBLOX_ROS_TSDF_SERVER_H_

#include <pcl/conversions.h>
#include <pcl/filters/filter.h>
#include <pcl/point_types.h>
#include <pcl_conversions/pcl_conversions.h>
#include <pcl_ros/point_cloud.h>
#include <ros/ros.h>
#include <sensor_msgs/PointCloud2.h>
#include <std_srvs/Empty.h>
#include <tf/transform_broadcaster.h>
#include <visualization_msgs/MarkerArray.h>
#include <memory>
#include <queue>
#include <string>

#include <voxblox/alignment/icp.h>
#include <voxblox/core/tsdf_map.h>
#include <voxblox/integrator/tsdf_integrator.h>
#include <voxblox/io/layer_io.h>
#include <voxblox/io/mesh_ply.h>
#include <voxblox/mesh/mesh_integrator.h>
#include <voxblox/utils/color_maps.h>
#include <voxblox_msgs/FilePath.h>
#include <voxblox_msgs/Mesh.h>

#include "voxblox_ros/mesh_vis.h"
#include "voxblox_ros/ptcloud_vis.h"
#include "voxblox_ros/transformer.h"

namespace voxblox {

constexpr float kDefaultMaxIntensity = 100.0;

class TsdfServer {
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW

  TsdfServer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private);
  TsdfServer(const ros::NodeHandle& nh, const ros::NodeHandle& nh_private,
             const TsdfMap::Config& config,
             const TsdfIntegratorBase::Config& integrator_config);
  virtual ~TsdfServer() {}
```

(continues on next page)

(continued from previous page)

```
  void getServerConfigFromRosParam(const ros::NodeHandle& nh_private);

  void insertPointcloud(const sensor_msgs::PointCloud2::Ptr& pointcloud);

  void insertFreespacePointcloud(
      const sensor_msgs::PointCloud2::Ptr& pointcloud);

  virtual void processPointCloudMessageAndInsert(
      const sensor_msgs::PointCloud2::Ptr& pointcloud_msg,
      const Transformation& T_G_C, const bool is_freespace_pointcloud);

  void integratePointcloud(const Transformation& T_G_C,
                           const Pointcloud& ptcloud_C, const Colors& colors,
                           const bool is_freespace_pointcloud = false);
  virtual void newPoseCallback(const Transformation& /*new_pose*/) {
    // Do nothing.
  }

  void publishAllUpdatedTsdfVoxels();
  void publishTsdfSurfacePoints();
  void publishTsdfOccupiedNodes();

  virtual void publishSlices();
  virtual void updateMesh();
  virtual bool generateMesh();
  // Publishes all available pointclouds.
  virtual void publishPointclouds();
  // Publishes the complete map
  virtual void publishMap(const bool reset_remote_map = false);
  virtual bool saveMap(const std::string& file_path);
  virtual bool loadMap(const std::string& file_path);

  bool clearMapCallback(std_srvs::Empty::Request& request,           // NOLINT
                        std_srvs::Empty::Response& response);       // NOLINT
  bool saveMapCallback(voxblox_msgs::FilePath::Request& request,    // NOLINT
                       voxblox_msgs::FilePath::Response& response); // NOLINT
  bool loadMapCallback(voxblox_msgs::FilePath::Request& request,    // NOLINT
                       voxblox_msgs::FilePath::Response& response); // NOLINT
  bool generateMeshCallback(std_srvs::Empty::Request& request,      // NOLINT
                            std_srvs::Empty::Response& response);   // NOLINT
  bool publishPointcloudsCallback(
      std_srvs::Empty::Request& request,                            // NOLINT
      std_srvs::Empty::Response& response);                         // NOLINT
  bool publishTsdfMapCallback(std_srvs::Empty::Request& request,    // NOLINT
                              std_srvs::Empty::Response& response); // NOLINT

  void updateMeshEvent(const ros::TimerEvent& event);

  std::shared_ptr<TsdfMap> getTsdfMapPtr() { return tsdf_map_; }

  double getSliceLevel() const { return slice_level_; }
  void setSliceLevel(double slice_level) { slice_level_ = slice_level; }

  bool setPublishSlices() const { return publish_slices_; }
  void setPublishSlices(const bool publish_slices) {
    publish_slices_ = publish_slices;
```

(continues on next page)

(continued from previous page)

```
  }

  void setWorldFrame(const std::string& world_frame) {
    world_frame_ = world_frame;
  }
  std::string getWorldFrame() const { return world_frame_; }

  virtual void clear();

  void tsdfMapCallback(const voxblox_msgs::Layer& layer_msg);

 protected:
  bool getNextPointcloudFromQueue(
      std::queue<sensor_msgs::PointCloud2::Ptr>* queue,
      sensor_msgs::PointCloud2::Ptr* pointcloud_msg, Transformation* T_G_C);

  ros::NodeHandle nh_;
  ros::NodeHandle nh_private_;

  bool verbose_;

  std::string world_frame_;
  std::string icp_corrected_frame_;
  std::string pose_corrected_frame_;

  double max_block_distance_from_body_;

  double slice_level_;

  bool use_freespace_pointcloud_;

  std::string mesh_filename_;
  ColorMode color_mode_;

  std::unique_ptr<ColorMap> color_map_;

  ros::Duration min_time_between_msgs_;

  bool publish_tsdf_info_;
  bool publish_slices_;
  bool publish_pointclouds_;
  bool publish_tsdf_map_;

  bool cache_mesh_;

  bool enable_icp_;
  bool accumulate_icp_corrections_;

  ros::Subscriber pointcloud_sub_;
  ros::Subscriber freespace_pointcloud_sub_;

  int pointcloud_queue_size_;

  // Publish markers for visualization.
  ros::Publisher mesh_pub_;
  ros::Publisher tsdf_pointcloud_pub_;
  ros::Publisher surface_pointcloud_pub_;
```

(continues on next page)

(continued from previous page)

```
  ros::Publisher tsdf_slice_pub_;
  ros::Publisher occupancy_marker_pub_;
  ros::Publisher icp_transform_pub_;

  ros::Publisher tsdf_map_pub_;

  ros::Subscriber tsdf_map_sub_;

  // Services.
  ros::ServiceServer generate_mesh_srv_;
  ros::ServiceServer clear_map_srv_;
  ros::ServiceServer save_map_srv_;
  ros::ServiceServer load_map_srv_;
  ros::ServiceServer publish_pointclouds_srv_;
  ros::ServiceServer publish_tsdf_map_srv_;

  tf::TransformBroadcaster tf_broadcaster_;

  // Timers.
  ros::Timer update_mesh_timer_;

  // Maps and integrators.
  std::shared_ptr<TsdfMap> tsdf_map_;
  std::unique_ptr<TsdfIntegratorBase> tsdf_integrator_;

  std::shared_ptr<ICP> icp_;

  // Mesh accessories.
  std::shared_ptr<MeshLayer> mesh_layer_;
  std::unique_ptr<MeshIntegrator<TsdfVoxel>> mesh_integrator_;
  voxblox_msgs::Mesh cached_mesh_msg_;

  Transformer transformer_;
  std::queue<sensor_msgs::PointCloud2::Ptr> pointcloud_queue_;
  std::queue<sensor_msgs::PointCloud2::Ptr> freespace_pointcloud_queue_;

  // Last message times for throttling input.
  ros::Time last_msg_time_ptcloud_;
  ros::Time last_msg_time_freespace_ptcloud_;

  Transformation icp_corrected_transform_;
};

}  // namespace voxblox

#endif  // VOXBLOX_ROS_TSDF_SERVER_H_
```

Includes

- memory
- pcl/conversions.h
- pcl/filters/filter.h
- pcl/point_types.h

- pcl_conversions/pcl_conversions.h
- pcl_ros/point_cloud.h
- queue (*File bucket_queue.h*)
- ros/ros.h
- sensor_msgs/PointCloud2.h
- std_srvs/Empty.h
- string
- tf/transform_broadcaster.h
- visualization_msgs/MarkerArray.h
- voxblox/alignment/icp.h (*File icp.h*)
- voxblox/core/tsdf_map.h (*File tsdf_map.h*)
- voxblox/integrator/tsdf_integrator.h (*File tsdf_integrator.h*)
- voxblox/io/layer_io.h (*File layer_io.h*)
- voxblox/io/mesh_ply.h (*File mesh_ply.h*)
- voxblox/mesh/mesh_integrator.h (*File mesh_integrator.h*)
- voxblox/utils/color_maps.h (*File color_maps.h*)
- voxblox_msgs/FilePath.h
- voxblox_msgs/Mesh.h
- voxblox_ros/mesh_vis.h (*File mesh_vis.h*)
- voxblox_ros/ptcloud_vis.h (*File ptcloud_vis.h*)
- voxblox_ros/transformer.h (*File transformer.h*)

Included By

- *File esdf_server.h*
- *File intensity_server.h*

Namespaces

- *Namespace voxblox*

Classes

- *Class TsdfServer*

File voxblox_mesh_display.h

Contents

- *Definition (voxblox_rviz_plugin/include/voxblox_rviz_plugin/voxblox_mesh_display.h)*
- *Includes*
- *Namespaces*
- *Classes*

Definition (voxblox_rviz_plugin/include/voxblox_rviz_plugin/voxblox_mesh_display.h)

Program Listing for File voxblox_mesh_display.h

Return to documentation for file (voxblox_rviz_plugin/include/voxblox_rviz_plugin/voxblox_mesh_display.h)

```
#ifndef VOXBLOX_RVIZ_PLUGIN_VOXBLOX_MESH_DISPLAY_H_
#define VOXBLOX_RVIZ_PLUGIN_VOXBLOX_MESH_DISPLAY_H_

#include <rviz/message_filter_display.h>
#include <voxblox_msgs/Mesh.h>
#include <memory>

#include "voxblox_rviz_plugin/voxblox_mesh_visual.h"

namespace voxblox_rviz_plugin {

class VoxbloxMeshVisual;

class VoxbloxMeshDisplay
    : public rviz::MessageFilterDisplay<voxblox_msgs::Mesh> {
  Q_OBJECT
 public:
  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
  VoxbloxMeshDisplay();
  virtual ~VoxbloxMeshDisplay();

 protected:
  virtual void onInitialize();

  virtual void reset();

 private:
  void processMessage(const voxblox_msgs::Mesh::ConstPtr& msg);

  std::unique_ptr<VoxbloxMeshVisual> visual_;
};

}  // namespace voxblox_rviz_plugin
```

(continues on next page)

(continued from previous page)

```
#endif  // VOXBLOX_RVIZ_PLUGIN_VOXBLOX_MESH_DISPLAY_H_
```

Includes

- memory
- rviz/message_filter_display.h
- voxblox_msgs/Mesh.h
- voxblox_rviz_plugin/voxblox_mesh_visual.h (*File voxblox_mesh_visual.h*)

Namespaces

- *Namespace voxblox_rviz_plugin*

Classes

- *Class VoxbloxMeshDisplay*

File voxblox_mesh_visual.h

Contents

- *Definition (voxblox_rviz_plugin/include/voxblox_rviz_plugin/voxblox_mesh_visual.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox_rviz_plugin/include/voxblox_rviz_plugin/voxblox_mesh_visual.h)

Program Listing for File voxblox_mesh_visual.h

Return to documentation for file (voxblox_rviz_plugin/include/voxblox_rviz_plugin/voxblox_mesh_visual.h)

```
#ifndef VOXBLOX_RVIZ_PLUGIN_VOXBLOX_MESH_VISUAL_H_
#define VOXBLOX_RVIZ_PLUGIN_VOXBLOX_MESH_VISUAL_H_

#include <OGRE/OgreManualObject.h>

#include <voxblox/core/block_hash.h>
```

(continues on next page)

(continued from previous page)

```
#include <voxblox_msgs/Mesh.h>

namespace voxblox_rviz_plugin {

class VoxbloxMeshVisual {
 public:
  VoxbloxMeshVisual(Ogre::SceneManager* scene_manager,
                    Ogre::SceneNode* parent_node);
  virtual ~VoxbloxMeshVisual();

  void setMessage(const voxblox_msgs::Mesh::ConstPtr& msg);

  void setFramePosition(const Ogre::Vector3& position);
  void setFrameOrientation(const Ogre::Quaternion& orientation);

 private:
  Ogre::SceneNode* frame_node_;
  Ogre::SceneManager* scene_manager_;

  unsigned int instance_number_;
  static unsigned int instance_counter_;

  voxblox::AnyIndexHashMapType<Ogre::ManualObject*>::type object_map_;
};

}  // namespace voxblox_rviz_plugin

#endif  // VOXBLOX_RVIZ_PLUGIN_VOXBLOX_MESH_VISUAL_H_
```

Includes

- OGRE/OgreManualObject.h
- voxblox/core/block_hash.h (*File block_hash.h*)
- voxblox_msgs/Mesh.h

Included By

- *File voxblox_mesh_display.h*

Namespaces

- *Namespace voxblox_rviz_plugin*

Classes

- *Class VoxbloxMeshVisual*

File voxel.h

Contents

- *Definition (voxblox/include/voxblox/core/voxel.h)*
- *Includes*
- *Included By*
- *Namespaces*
- *Classes*

Definition (voxblox/include/voxblox/core/voxel.h)

Program Listing for File voxel.h

Return to documentation for file (voxblox/include/voxblox/core/voxel.h)

```
#ifndef VOXBLOX_CORE_VOXEL_H_
#define VOXBLOX_CORE_VOXEL_H_

#include <cstdint>
#include <string>

#include "voxblox/core/color.h"
#include "voxblox/core/common.h"

namespace voxblox {

struct TsdfVoxel {
  float distance = 0.0f;
  float weight = 0.0f;
  Color color;
};

struct EsdfVoxel {
  float distance = 0.0f;

  bool observed = false;
  bool hallucinated = false;
  bool in_queue = false;
  bool fixed = false;

  Eigen::Vector3i parent = Eigen::Vector3i::Zero();

  EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

struct OccupancyVoxel {
  float probability_log = 0.0f;
  bool observed = false;
};
```

(continues on next page)

(continued from previous page)

```
struct IntensityVoxel {
  float intensity = 0.0f;
  float weight = 0.0f;
};

namespace voxel_types {
const std::string kNotSerializable = "not_serializable";
const std::string kTsdf = "tsdf";
const std::string kEsdf = "esdf";
const std::string kOccupancy = "occupancy";
const std::string kIntensity = "intensity";
}  // namespace voxel_types

template <typename Type>
std::string getVoxelType() {
  return voxel_types::kNotSerializable;
}

template <>
inline std::string getVoxelType<TsdfVoxel>() {
  return voxel_types::kTsdf;
}

template <>
inline std::string getVoxelType<EsdfVoxel>() {
  return voxel_types::kEsdf;
}

template <>
inline std::string getVoxelType<OccupancyVoxel>() {
  return voxel_types::kOccupancy;
}

template <>
inline std::string getVoxelType<IntensityVoxel>() {
  return voxel_types::kIntensity;
}

}  // namespace voxblox

#endif  // VOXBLOX_CORE_VOXEL_H_
```

Includes

- cstdint
- string
- voxblox/core/color.h (*File color.h*)
- voxblox/core/common.h (*File common.h*)

Included By

- *File esdf_map.h*

- *File layer.h*
- *File layer_inl.h*
- *File occupancy_map.h*
- *File tsdf_map.h*
- *File esdf_integrator.h*
- *File esdf_occ_integrator.h*
- *File intensity_integrator.h*
- *File merge_integration.h*
- *File occupancy_integrator.h*
- *File tsdf_integrator.h*
- *File interpolator.h*
- *File mesh_integrator.h*
- *File objects.h*
- *File simulation_world.h*
- *File layer_test_utils.h*
- *File distance_utils.h*
- *File evaluation_utils.h*
- *File layer_utils.h*
- *File meshing_utils.h*
- *File planning_utils.h*
- *File planning_utils_inl.h*
- *File voxel_utils.h*
- *File intensity_server.h*
- *File ptcloud_vis.h*

Namespaces

- *Namespace voxblox*
- *Namespace voxblox::voxel_types*

Classes

- *Struct EsdfVoxel*
- *Struct IntensityVoxel*
- *Struct OccupancyVoxel*
- *Struct TsdfVoxel*

File voxel_utils.h

Contents

- *Definition (voxblox/include/voxblox/utils/voxel_utils.h)*
- *Includes*
- *Included By*
- *Namespaces*

Definition (voxblox/include/voxblox/utils/voxel_utils.h)

Program Listing for File voxel_utils.h

Return to documentation for file (voxblox/include/voxblox/utils/voxel_utils.h)

```
#ifndef VOXBLOX_UTILS_VOXEL_UTILS_H_
#define VOXBLOX_UTILS_VOXEL_UTILS_H_

#include "voxblox/core/color.h"
#include "voxblox/core/common.h"
#include "voxblox/core/voxel.h"

namespace voxblox {
template <typename VoxelType>
void mergeVoxelAIntoVoxelB(const VoxelType& voxel_A, VoxelType* voxel_B);

template <>
void mergeVoxelAIntoVoxelB(const TsdfVoxel& voxel_A, TsdfVoxel* voxel_B);

template <>
void mergeVoxelAIntoVoxelB(const EsdfVoxel& voxel_A, EsdfVoxel* voxel_B);

template <>
void mergeVoxelAIntoVoxelB(const OccupancyVoxel& voxel_A,
                           OccupancyVoxel* voxel_B);

}  // namespace voxblox

#endif  // VOXBLOX_UTILS_VOXEL_UTILS_H_
```

Includes

- voxblox/core/color.h (*File color.h*)
- voxblox/core/common.h (*File common.h*)
- voxblox/core/voxel.h (*File voxel.h*)

Included By

- *File block_inl.h*

Namespaces

- *Namespace voxblox*

CHAPTER 11

Paper and Video

A video showing sample output from voxblox can be seen here. A video of voxblox being used for online planning on-board a multicopter can be seen here.

If using voxblox for scientific publications, please cite the following paper, available here:

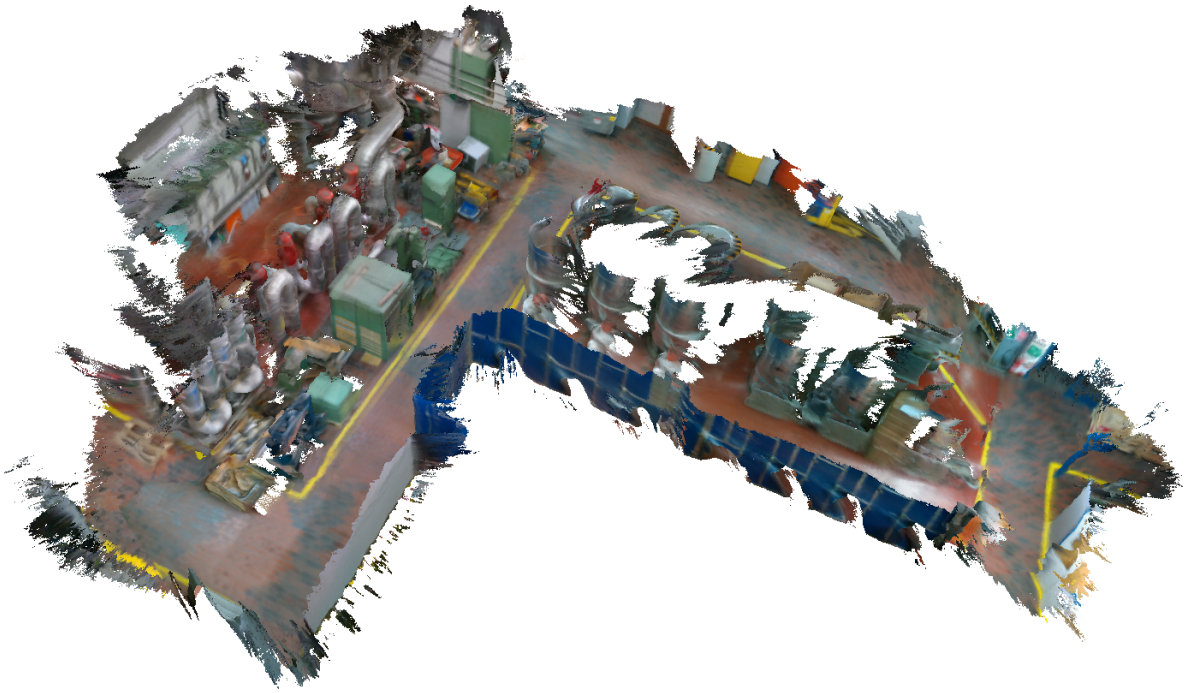
Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan Nieto, and Roland Siegwart, "**Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning**", in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.:

```
@inproceedings{oleynikova2017voxblox,
  author={Oleynikova, Helen and Taylor, Zachary and Fehr, Marius and Siegwart, Roland␣
↪and  Nieto, Juan},
  booktitle={IEEE/RSJ International Conference on Intelligent Robots and Systems␣
↪(IROS)},
  title={Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV␣
↪Planning},
  year={2017}
}
```

CHAPTER 12

Credits

This library was written primarily by Helen Oleynikova and Marius Fehr, with significant contributions from Zachary Taylor, Alexander Millane, and others. The marching cubes meshing and ROS mesh generation were taken or heavily derived from open_chisel. We've retained the copyright headers for the relevant files.



Index

V

voxblox::aligned_shared (C++ function), 118
voxblox::AlignedLayerAndErrorLayer (C++ type), 152
voxblox::AlignedLayerAndErrorLayers (C++ type), 152
voxblox::AnyIndex (C++ type), 153
voxblox::AnyIndexHash (C++ class), 49
voxblox::AnyIndexHash::operator() (C++ function), 49
voxblox::AnyIndexHash::sl (C++ member), 49
voxblox::AnyIndexHash::sl2 (C++ member), 49
voxblox::AnyIndexHashMapType (C++ class), 49
voxblox::AnyIndexHashMapType::type (C++ type), 50
voxblox::ApproxHashArray (C++ class), 60
voxblox::ApproxHashArray::get (C++ function), 60
voxblox::ApproxHashSet (C++ class), 61
voxblox::ApproxHashSet::ApproxHashSet (C++ function), 61
voxblox::ApproxHashSet::isHashCurrentlyPresent (C++ function), 61
voxblox::ApproxHashSet::replaceHash (C++ function), 61
voxblox::ApproxHashSet::resetApproxSet (C++ function), 61
voxblox::Block (C++ class), 62
voxblox::Block::~Block (C++ function), 62
voxblox::Block::Block (C++ function), 62
voxblox::Block::block_index (C++ function), 63
voxblox::Block::block_size (C++ function), 63
voxblox::Block::computeCoordinatesFromLinearIndex (C++ function), 62
voxblox::Block::computeCoordinatesFromVoxelIndex (C++ function), 62
voxblox::Block::computeLinearIndexFromCoordinates (C++ function), 62
voxblox::Block::computeLinearIndexFromVoxelIndex (C++ function), 62
voxblox::Block::computeTruncatedVoxelIndexFromCoordinates (C++ function), 62
voxblox::Block::computeVoxelIndexFromCoordinates (C++ function), 62
voxblox::Block::computeVoxelIndexFromLinearIndex (C++ function), 62
voxblox::Block::ConstPtr (C++ type), 62
voxblox::Block::deserializeFromIntegers (C++ function), 64
voxblox::Block::getMemorySize (C++ function), 64
voxblox::Block::getProto (C++ function), 63
voxblox::Block::getVoxelByCoordinates (C++ function), 62, 63
voxblox::Block::getVoxelByLinearIndex (C++ function), 62, 63
voxblox::Block::getVoxelByVoxelIndex (C++ function), 62, 63
voxblox::Block::getVoxelPtrByCoordinates (C++ function), 63
voxblox::Block::has_data (C++ function), 63
voxblox::Block::has_data_ (C++ member), 64
voxblox::Block::isValidLinearIndex (C++ function), 63
voxblox::Block::isValidVoxelIndex (C++ function), 63
voxblox::Block::mergeBlock (C++ function), 64
voxblox::Block::num_voxels (C++ function), 63
voxblox::Block::num_voxels_ (C++ member), 64
voxblox::Block::origin (C++ function), 63
voxblox::Block::Ptr (C++ type), 62
voxblox::Block::serializeToIntegers (C++ function), 63
voxblox::Block::set_has_data (C++ function), 63
voxblox::Block::set_updated (C++ function), 63
voxblox::Block::setOrigin (C++ function), 63
voxblox::Block::updated (C++ function), 63
voxblox::Block::voxel_size (C++ function), 63
voxblox::Block::voxel_size_inv (C++ function), 63
voxblox::Block::voxels_ (C++ member), 64
voxblox::Block::voxels_per_side (C++ function), 63
voxblox::BlockIndex (C++ type), 153
voxblox::BlockIndexList (C++ type), 153
voxblox::BucketQueue (C++ class), 64
voxblox::BucketQueue::BucketQueue (C++ function), 64
voxblox::BucketQueue::clear (C++ function), 64
voxblox::BucketQueue::empty (C++ function), 64
voxblox::BucketQueue::front (C++ function), 64

voxblox::BucketQueue::pop (C++ function), 64
voxblox::BucketQueue::push (C++ function), 64
voxblox::BucketQueue::setNumBuckets (C++ function), 64
voxblox::CameraModel (C++ class), 65
voxblox::CameraModel::~CameraModel (C++ function), 65
voxblox::CameraModel::CameraModel (C++ function), 65
voxblox::CameraModel::getAabb (C++ function), 65
voxblox::CameraModel::getBodyPose (C++ function), 65
voxblox::CameraModel::getBoundingLines (C++ function), 65
voxblox::CameraModel::getBoundingPlanes (C++ function), 65
voxblox::CameraModel::getCameraPose (C++ function), 65
voxblox::CameraModel::getFarPlanePoints (C++ function), 65
voxblox::CameraModel::isPointInView (C++ function), 65
voxblox::CameraModel::setBodyPose (C++ function), 65
voxblox::CameraModel::setCameraPose (C++ function), 65
voxblox::CameraModel::setExtrinsics (C++ function), 65
voxblox::CameraModel::setIntrinsicsFromFocalLength (C++ function), 65
voxblox::CameraModel::setIntrinsicsFromFoV (C++ function), 65
voxblox::castRay (C++ function), 118
voxblox::Color (C++ class), 50
voxblox::Color::a (C++ member), 50
voxblox::Color::b (C++ member), 50
voxblox::Color::Black (C++ function), 50
voxblox::Color::blendTwoColors (C++ function), 50
voxblox::Color::Blue (C++ function), 50
voxblox::Color::Color (C++ function), 50
voxblox::Color::g (C++ member), 50
voxblox::Color::Gray (C++ function), 50
voxblox::Color::Green (C++ function), 50
voxblox::Color::Orange (C++ function), 50
voxblox::Color::Pink (C++ function), 51
voxblox::Color::Purple (C++ function), 50
voxblox::Color::r (C++ member), 50
voxblox::Color::Red (C++ function), 50
voxblox::Color::Teal (C++ function), 50
voxblox::Color::White (C++ function), 50
voxblox::Color::Yellow (C++ function), 50
voxblox::ColorMap (C++ class), 66
voxblox::ColorMap::~ColorMap (C++ function), 66
voxblox::ColorMap::colorLookup (C++ function), 66
voxblox::ColorMap::ColorMap (C++ function), 66
voxblox::ColorMap::max_value_ (C++ member), 66
voxblox::ColorMap::min_value_ (C++ member), 66
voxblox::ColorMap::setMaxValue (C++ function), 66
voxblox::ColorMap::setMinValue (C++ function), 66
voxblox::ColorMode (C++ type), 116
voxblox::colorMsgToVoxblox (C++ function), 118
voxblox::Colors (C++ type), 154
voxblox::colorVoxbloxToMsg (C++ function), 118
voxblox::Connectivity (C++ type), 116
voxblox::createColorPointcloudFromLayer (C++ function), 119
voxblox::createDistancePointcloudFromEsdfLayer (C++ function), 120
voxblox::createDistancePointcloudFromEsdfLayerSlice (C++ function), 120
voxblox::createDistancePointcloudFromTsdfLayer (C++ function), 120
voxblox::createDistancePointcloudFromTsdfLayerSlice (C++ function), 120
voxblox::createFreePointcloudFromEsdfLayer (C++ function), 121
voxblox::createIntensityPointcloudFromIntensityLayer (C++ function), 121
voxblox::createOccupancyBlocksFromLayer (C++ function), 121
voxblox::createOccupancyBlocksFromOccupancyLayer (C++ function), 121
voxblox::createOccupancyBlocksFromTsdfLayer (C++ function), 122
voxblox::createPointcloudFromTsdfLayer (C++ function), 122
voxblox::createSurfaceDistancePointcloudFromTsdfLayer (C++ function), 122
voxblox::createSurfacePointcloudFromTsdfLayer (C++ function), 122
voxblox::Cube (C++ class), 66
voxblox::Cube::Cube (C++ function), 67
voxblox::Cube::getDistanceToPoint (C++ function), 67
voxblox::Cube::getRayIntersection (C++ function), 67
voxblox::Cube::size_ (C++ member), 67
voxblox::Cylinder (C++ class), 67
voxblox::Cylinder::Cylinder (C++ function), 67
voxblox::Cylinder::getDistanceToPoint (C++ function), 67
voxblox::Cylinder::getRayIntersection (C++ function), 67
voxblox::Cylinder::height_ (C++ member), 68
voxblox::Cylinder::radius_ (C++ member), 68
voxblox::deserializeMsgToLayer (C++ function), 123
voxblox::EsdfIntegrator (C++ class), 68
voxblox::EsdfIntegrator::addNewRobotPosition (C++ function), 68
voxblox::EsdfIntegrator::clear (C++ function), 69
voxblox::EsdfIntegrator::Config (C++ class), 51, 69
voxblox::EsdfIntegrator::Config::add_occupied_crust

(C++ member), 51, 70
voxblox::EsdfIntegrator::Config::clear_sphere_radius (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::default_distance_m (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::full_euclidean_distance (C++ member), 51, 69
voxblox::EsdfIntegrator::Config::max_distance_m (C++ member), 51, 69
voxblox::EsdfIntegrator::Config::min_diff_m (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::min_distance_m (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::min_weight (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::multi_queue (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::num_buckets (C++ member), 51, 70
voxblox::EsdfIntegrator::Config::occupied_sphere_radius (C++ member), 51, 70
voxblox::EsdfIntegrator::config_ (C++ member), 69
voxblox::EsdfIntegrator::esdf_layer_ (C++ member), 69
voxblox::EsdfIntegrator::EsdfIntegrator (C++ function), 68
voxblox::EsdfIntegrator::getEsdfMaxDistance (C++ function), 69
voxblox::EsdfIntegrator::getFullEuclidean (C++ function), 69
voxblox::EsdfIntegrator::isFixed (C++ function), 69
voxblox::EsdfIntegrator::open_ (C++ member), 69
voxblox::EsdfIntegrator::processOpenSet (C++ function), 68
voxblox::EsdfIntegrator::processRaiseSet (C++ function), 68
voxblox::EsdfIntegrator::raise_ (C++ member), 69
voxblox::EsdfIntegrator::setEsdfMaxDistance (C++ function), 69
voxblox::EsdfIntegrator::setFullEuclidean (C++ function), 69
voxblox::EsdfIntegrator::tsdf_layer_ (C++ member), 69
voxblox::EsdfIntegrator::updated_blocks_ (C++ member), 69
voxblox::EsdfIntegrator::updateFromTsdfBlocks (C++ function), 68
voxblox::EsdfIntegrator::updateFromTsdfLayer (C++ function), 68
voxblox::EsdfIntegrator::updateFromTsdfLayerBatch (C++ function), 68
voxblox::EsdfIntegrator::updateVoxelFromNeighbors (C++ function), 69
voxblox::EsdfIntegrator::voxel_size_ (C++ member), 69
voxblox::EsdfIntegrator::voxels_per_side_ (C++ member), 69
voxblox::EsdfMap (C++ class), 70
voxblox::EsdfMap::~EsdfMap (C++ function), 71
voxblox::EsdfMap::batchGetDistanceAndGradientAtPosition (C++ function), 71
voxblox::EsdfMap::batchGetDistanceAtPosition (C++ function), 71
voxblox::EsdfMap::batchIsObserved (C++ function), 71
voxblox::EsdfMap::block_size (C++ function), 71
voxblox::EsdfMap::block_size_ (C++ member), 72
voxblox::EsdfMap::Config (C++ class), 52, 72
voxblox::EsdfMap::Config::esdf_voxel_size (C++ member), 52, 72
voxblox::EsdfMap::Config::esdf_voxels_per_side (C++ member), 52, 72
voxblox::EsdfMap::coordPlaneSliceGetCount (C++ function), 72
voxblox::EsdfMap::coordPlaneSliceGetDistance (C++ function), 72
voxblox::EsdfMap::EigenDRef (C++ type), 71
voxblox::EsdfMap::EigenDStride (C++ type), 71
voxblox::EsdfMap::esdf_layer_ (C++ member), 72
voxblox::EsdfMap::EsdfMap (C++ function), 71
voxblox::EsdfMap::getDistanceAndGradientAtPosition (C++ function), 71
voxblox::EsdfMap::getDistanceAtPosition (C++ function), 71
voxblox::EsdfMap::getEsdfLayer (C++ function), 71
voxblox::EsdfMap::getEsdfLayerPtr (C++ function), 71
voxblox::EsdfMap::interpolator_ (C++ member), 72
voxblox::EsdfMap::isObserved (C++ function), 71
voxblox::EsdfMap::Ptr (C++ type), 71
voxblox::EsdfMap::voxel_size (C++ function), 71
voxblox::EsdfOccIntegrator (C++ class), 72
voxblox::EsdfOccIntegrator::Config (C++ class), 52, 73
voxblox::EsdfOccIntegrator::Config::default_distance_m (C++ member), 52, 73
voxblox::EsdfOccIntegrator::Config::max_distance_m (C++ member), 52, 73
voxblox::EsdfOccIntegrator::Config::num_buckets (C++ member), 52, 73
voxblox::EsdfOccIntegrator::config_ (C++ member), 73
voxblox::EsdfOccIntegrator::esdf_layer_ (C++ member), 73
voxblox::EsdfOccIntegrator::esdf_voxel_size_ (C++ member), 73
voxblox::EsdfOccIntegrator::esdf_voxels_per_side_ (C++ member), 73
voxblox::EsdfOccIntegrator::EsdfOccIntegrator (C++ function), 73
voxblox::EsdfOccIntegrator::getNeighbor (C++ function), 73
voxblox::EsdfOccIntegrator::getNeighborsAndDistances (C++ function), 73
voxblox::EsdfOccIntegrator::occ_layer_ (C++ member),

73
voxblox::EsdfOccIntegrator::open_ (C++ member), 73
voxblox::EsdfOccIntegrator::processOpenSet (C++ function), 73
voxblox::EsdfOccIntegrator::raise_ (C++ member), 73
voxblox::EsdfOccIntegrator::updateFromOccBlocks (C++ function), 73
voxblox::EsdfOccIntegrator::updateFromOccLayerBatch (C++ function), 73
voxblox::EsdfServer (C++ class), 74
voxblox::EsdfServer::~EsdfServer (C++ function), 74
voxblox::EsdfServer::clear (C++ function), 75
voxblox::EsdfServer::clear_sphere_for_planning_ (C++ member), 75
voxblox::EsdfServer::disableIncrementalUpdate (C++ function), 75
voxblox::EsdfServer::enableIncrementalUpdate (C++ function), 75
voxblox::EsdfServer::esdf_integrator_ (C++ member), 76
voxblox::EsdfServer::esdf_map_ (C++ member), 76
voxblox::EsdfServer::esdf_map_pub_ (C++ member), 75
voxblox::EsdfServer::esdf_map_sub_ (C++ member), 75
voxblox::EsdfServer::esdf_pointcloud_pub_ (C++ member), 75
voxblox::EsdfServer::esdf_slice_pub_ (C++ member), 75
voxblox::EsdfServer::esdfMapCallback (C++ function), 75
voxblox::EsdfServer::EsdfServer (C++ function), 74
voxblox::EsdfServer::generate_esdf_srv_ (C++ member), 75
voxblox::EsdfServer::generateEsdfCallback (C++ function), 74
voxblox::EsdfServer::getClearSphere (C++ function), 75
voxblox::EsdfServer::getEsdfMapPtr (C++ function), 75
voxblox::EsdfServer::getEsdfMaxDistance (C++ function), 75
voxblox::EsdfServer::getTraversabilityRadius (C++ function), 75
voxblox::EsdfServer::incremental_update_ (C++ member), 76
voxblox::EsdfServer::loadMap (C++ function), 74
voxblox::EsdfServer::newPoseCallback (C++ function), 74
voxblox::EsdfServer::publish_esdf_map_ (C++ member), 75
voxblox::EsdfServer::publish_traversable_ (C++ member), 75
voxblox::EsdfServer::publishAllUpdatedEsdfVoxels (C++ function), 74
voxblox::EsdfServer::publishMap (C++ function), 74
voxblox::EsdfServer::publishPointclouds (C++ function), 74
voxblox::EsdfServer::publishSlices (C++ function), 74
voxblox::EsdfServer::publishTraversable (C++ function), 74
voxblox::EsdfServer::saveMap (C++ function), 74
voxblox::EsdfServer::setClearSphere (C++ function), 75
voxblox::EsdfServer::setEsdfMaxDistance (C++ function), 75
voxblox::EsdfServer::setTraversabilityRadius (C++ function), 75
voxblox::EsdfServer::setupRos (C++ function), 75
voxblox::EsdfServer::traversability_radius_ (C++ member), 75
voxblox::EsdfServer::traversable_pub_ (C++ member), 75
voxblox::EsdfServer::updateEsdf (C++ function), 74
voxblox::EsdfServer::updateEsdfBatch (C++ function), 74
voxblox::EsdfServer::updateMesh (C++ function), 74
voxblox::EsdfVoxel (C++ class), 53
voxblox::EsdfVoxel::distance (C++ member), 53
voxblox::EsdfVoxel::fixed (C++ member), 53
voxblox::EsdfVoxel::hallucinated (C++ member), 53
voxblox::EsdfVoxel::in_queue (C++ member), 53
voxblox::EsdfVoxel::observed (C++ member), 53
voxblox::EsdfVoxel::parent (C++ member), 53
voxblox::evaluateLayerRmseAtPoses (C++ function), 123, 124
voxblox::FastTsdfIntegrator (C++ class), 76
voxblox::FastTsdfIntegrator::FastTsdfIntegrator (C++ function), 76
voxblox::FastTsdfIntegrator::integrateFunction (C++ function), 76
voxblox::FastTsdfIntegrator::integratePointCloud (C++ function), 76
voxblox::fillMarkerWithMesh (C++ function), 124
voxblox::fillPointcloudWithMesh (C++ function), 124
voxblox::FloatingPoint (C++ type), 154
voxblox::generateVoxbloxMeshMsg (C++ function), 124
voxblox::getBlockAndVoxelIndexFromGlobalVoxelIndex (C++ function), 125
voxblox::getBlockIndexFromGlobalVoxelIndex (C++ function), 125
voxblox::getCenterPointFromGridIndex (C++ function), 125
voxblox::getEsdfIntegratorConfigFromRosParam (C++ function), 125
voxblox::getEsdfMapConfigFromRosParam (C++ function), 125
voxblox::getGlobalVoxelIndexFromBlockAndVoxelIndex (C++ function), 126
voxblox::getGridIndexFromOriginPoint (C++ function), 126
voxblox::getGridIndexFromPoint (C++ function), 126
voxblox::getHierarchicalIndexAlongRay (C++ function), 127
voxblox::getICPConfigFromRosParam (C++ function),

127
voxblox::getLocalFromGlobalVoxelIndex (C++ function), 127
voxblox::getOriginPointFromGridIndex (C++ function), 127
voxblox::getSurfaceDistanceAlongRay (C++ function), 128
voxblox::getTsdfIntegratorConfigFromRosParam (C++ function), 128
voxblox::getTsdfMapConfigFromRosParam (C++ function), 128
voxblox::getVertexColor (C++ function), 128
voxblox::getVoxelType (C++ function), 128
voxblox::getVoxelType<EsdfVoxel> (C++ function), 129
voxblox::getVoxelType<IntensityVoxel> (C++ function), 129
voxblox::getVoxelType<OccupancyVoxel> (C++ function), 129
voxblox::getVoxelType<TsdfVoxel> (C++ function), 129
voxblox::GlobalIndex (C++ type), 154
voxblox::GlobalIndexVector (C++ type), 154
voxblox::grayColorMap (C++ function), 129
voxblox::GrayscaleColorMap (C++ class), 77
voxblox::GrayscaleColorMap::colorLookup (C++ function), 77
voxblox::heightColorFromVertex (C++ function), 130
voxblox::HierarchicalIndex (C++ type), 154
voxblox::HierarchicalIndexMap (C++ type), 154
voxblox::HierarchicalIndexSet (C++ type), 155
voxblox::ICP (C++ class), 77
voxblox::ICP::Config (C++ class), 53, 78
voxblox::ICP::Config::inital_rotation_weighting (C++ member), 54, 78
voxblox::ICP::Config::inital_translation_weighting (C++ member), 53, 78
voxblox::ICP::Config::min_match_ratio (C++ member), 53, 78
voxblox::ICP::Config::mini_batch_size (C++ member), 53, 78
voxblox::ICP::Config::num_threads (C++ member), 54, 78
voxblox::ICP::Config::refine_roll_pitch (C++ member), 53, 78
voxblox::ICP::Config::subsample_keep_ratio (C++ member), 53, 78
voxblox::ICP::ICP (C++ function), 77
voxblox::ICP::refiningRollPitch (C++ function), 78
voxblox::ICP::runICP (C++ function), 78
voxblox::IndexElement (C++ type), 155
voxblox::IndexSet (C++ type), 155
voxblox::IndexVector (C++ type), 155
voxblox::IntensityIntegrator (C++ class), 78
voxblox::IntensityIntegrator::addIntensityBearingVectors (C++ function), 79
voxblox::IntensityIntegrator::getMaxDistance (C++ function), 79
voxblox::IntensityIntegrator::IntensityIntegrator (C++ function), 79
voxblox::IntensityIntegrator::setMaxDistance (C++ function), 79
voxblox::IntensityServer (C++ class), 79
voxblox::IntensityServer::~IntensityServer (C++ function), 79
voxblox::IntensityServer::color_map_ (C++ member), 80
voxblox::IntensityServer::focal_length_px_ (C++ member), 79
voxblox::IntensityServer::intensity_image_sub_ (C++ member), 79
voxblox::IntensityServer::intensity_integrator_ (C++ member), 80
voxblox::IntensityServer::intensity_layer_ (C++ member), 80
voxblox::IntensityServer::intensity_mesh_pub_ (C++ member), 79
voxblox::IntensityServer::intensity_pointcloud_pub_ (C++ member), 79
voxblox::IntensityServer::intensityImageCallback (C++ function), 79
voxblox::IntensityServer::IntensityServer (C++ function), 79
voxblox::IntensityServer::publishPointclouds (C++ function), 79
voxblox::IntensityServer::subsample_factor_ (C++ member), 80
voxblox::IntensityServer::updateMesh (C++ function), 79
voxblox::IntensityVoxel (C++ class), 54
voxblox::IntensityVoxel::intensity (C++ member), 54
voxblox::IntensityVoxel::weight (C++ member), 54
voxblox::InteractiveSlider (C++ class), 80
voxblox::InteractiveSlider::~InteractiveSlider (C++ function), 80
voxblox::InteractiveSlider::InteractiveSlider (C++ function), 80
voxblox::InterpIndexes (C++ type), 155
voxblox::Interpolator (C++ class), 80
voxblox::Interpolator::getAdaptiveDistanceAndGradient (C++ function), 81
voxblox::Interpolator::getDistance (C++ function), 80
voxblox::Interpolator::getGradient (C++ function), 80
voxblox::Interpolator::getNearestDistanceAndWeight (C++ function), 81
voxblox::Interpolator::getVoxel (C++ function), 81
voxblox::Interpolator::Interpolator (C++ function), 80
voxblox::Interpolator::Ptr (C++ type), 80
voxblox::InterpTable (C++ type), 155
voxblox::InterpVector (C++ type), 155
voxblox::InverseGrayscaleColorMap (C++ class), 81
voxblox::InverseGrayscaleColorMap::colorLookup (C++

function), 81
voxblox::InverseRainbowColorMap (C++ class), 81
voxblox::InverseRainbowColorMap::colorLookup (C++ function), 82
voxblox::io::convertVoxelGridToPointCloud (C++ function), 130, 131
voxblox::io::getColorFromVoxel (C++ function), 131
voxblox::io::kSdfColoredDistanceField (C++ enumerator), 116
voxblox::io::kSdfIsosurface (C++ enumerator), 116
voxblox::io::kSdfIsosurfaceConnected (C++ enumerator), 116
voxblox::io::LoadBlocksFromFile (C++ function), 132
voxblox::io::LoadLayer (C++ function), 132
voxblox::io::outputLayerAsPly (C++ function), 133
voxblox::io::PlyOutputTypes (C++ type), 116
voxblox::io::PlyWriter (C++ class), 82
voxblox::io::PlyWriter::~PlyWriter (C++ function), 82
voxblox::io::PlyWriter::addVerticesWithProperties (C++ function), 82
voxblox::io::PlyWriter::closeFile (C++ function), 82
voxblox::io::PlyWriter::PlyWriter (C++ function), 82
voxblox::io::PlyWriter::writeHeader (C++ function), 82
voxblox::io::PlyWriter::writeVertex (C++ function), 82
voxblox::io::SaveLayer (C++ function), 133
voxblox::io::SaveLayerSubset (C++ function), 133
voxblox::IronbowColorMap (C++ class), 82
voxblox::IronbowColorMap::colorLookup (C++ function), 83
voxblox::IronbowColorMap::increment_ (C++ member), 83
voxblox::IronbowColorMap::IronbowColorMap (C++ function), 83
voxblox::IronbowColorMap::palette_colors_ (C++ member), 83
voxblox::isPowerOfTwo (C++ function), 133
voxblox::kColor (C++ enumerator), 116
voxblox::kDefaultMaxIntensity (C++ member), 150
voxblox::kEighteen (C++ enumerator), 116
voxblox::kEpsilon (C++ member), 150
voxblox::kFast (C++ enumerator), 117
voxblox::kFloatEpsilon (C++ member), 151
voxblox::kGray (C++ enumerator), 116
voxblox::kHeight (C++ enumerator), 116
voxblox::kLambert (C++ enumerator), 116
voxblox::kLambertColor (C++ enumerator), 116
voxblox::kMerge (C++ enumerator), 117
voxblox::kMerged (C++ enumerator), 117
voxblox::kNormals (C++ enumerator), 116
voxblox::kNumTsdfIntegratorTypes (C++ member), 151
voxblox::kReset (C++ enumerator), 117
voxblox::kSimple (C++ enumerator), 117
voxblox::kSix (C++ enumerator), 116
voxblox::kTsdfIntegratorTypeNames (C++ member), 151
voxblox::kTwentySix (C++ enumerator), 116
voxblox::kUnitCubeDiagonalLength (C++ member), 151
voxblox::kUpdate (C++ enumerator), 117
voxblox::Label (C++ type), 156
voxblox::LabelConfidence (C++ type), 156
voxblox::Labels (C++ type), 156
voxblox::lambertColorFromColorAndNormal (C++ function), 134
voxblox::lambertColorFromNormal (C++ function), 134
voxblox::lambertShading (C++ function), 134
voxblox::Layer (C++ class), 83
voxblox::Layer::~Layer (C++ function), 83
voxblox::Layer::addBlockFromProto (C++ function), 85
voxblox::Layer::allocateBlockPtrByCoordinates (C++ function), 84
voxblox::Layer::allocateBlockPtrByIndex (C++ function), 84
voxblox::Layer::allocateNewBlock (C++ function), 84
voxblox::Layer::allocateNewBlockByCoordinates (C++ function), 84
voxblox::Layer::block_map_ (C++ member), 85
voxblox::Layer::block_size (C++ function), 84
voxblox::Layer::block_size_ (C++ member), 85
voxblox::Layer::block_size_inv (C++ function), 84
voxblox::Layer::block_size_inv_ (C++ member), 85
voxblox::Layer::BlockHashMap (C++ type), 83
voxblox::Layer::BlockMapPair (C++ type), 83
voxblox::Layer::BlockMergingStrategy (C++ type), 83
voxblox::Layer::BlockType (C++ type), 83
voxblox::Layer::computeBlockIndexFromCoordinates (C++ function), 84
voxblox::Layer::getAllAllocatedBlocks (C++ function), 84
voxblox::Layer::getAllUpdatedBlocks (C++ function), 84
voxblox::Layer::getBlockByIndex (C++ function), 84
voxblox::Layer::getBlockPtrByCoordinates (C++ function), 84
voxblox::Layer::getBlockPtrByIndex (C++ function), 84
voxblox::Layer::getMemorySize (C++ function), 85
voxblox::Layer::getNumberOfAllocatedBlocks (C++ function), 84
voxblox::Layer::getProto (C++ function), 85
voxblox::Layer::getType (C++ function), 85
voxblox::Layer::getVoxelPtrByCoordinates (C++ function), 84
voxblox::Layer::getVoxelPtrByGlobalIndex (C++ function), 84
voxblox::Layer::hasBlock (C++ function), 84
voxblox::Layer::insertBlock (C++ function), 84
voxblox::Layer::isCompatible (C++ function), 85
voxblox::Layer::kDiscard (C++ enumerator), 83
voxblox::Layer::kMerge (C++ enumerator), 83
voxblox::Layer::kProhibit (C++ enumerator), 83

voxblox::Layer::kReplace (C++ enumerator), 83
voxblox::Layer::Layer (C++ function), 83
voxblox::Layer::Ptr (C++ type), 83
voxblox::Layer::removeAllBlocks (C++ function), 84
voxblox::Layer::removeBlock (C++ function), 84
voxblox::Layer::removeBlockByCoordinates (C++ function), 84
voxblox::Layer::removeDistantBlocks (C++ function), 84
voxblox::Layer::saveSubsetToFile (C++ function), 85
voxblox::Layer::saveToFile (C++ function), 85
voxblox::Layer::voxel_size (C++ function), 84
voxblox::Layer::voxel_size_ (C++ member), 85
voxblox::Layer::voxel_size_inv (C++ function), 85
voxblox::Layer::voxel_size_inv_ (C++ member), 85
voxblox::Layer::voxels_per_side (C++ function), 85
voxblox::Layer::voxels_per_side_ (C++ member), 85
voxblox::Layer::voxels_per_side_inv (C++ function), 85
voxblox::Layer::voxels_per_side_inv_ (C++ member), 85
voxblox::logOddsFromProbability (C++ function), 134
voxblox::LongIndex (C++ type), 156
voxblox::LongIndexElement (C++ type), 156
voxblox::LongIndexHash (C++ class), 54
voxblox::LongIndexHash::operator() (C++ function), 54
voxblox::LongIndexHash::sl (C++ member), 54
voxblox::LongIndexHash::sl2 (C++ member), 54
voxblox::LongIndexHashMapType (C++ class), 55
voxblox::LongIndexHashMapType::type (C++ type), 55
voxblox::LongIndexSet (C++ type), 156
voxblox::LongIndexVector (C++ type), 157
voxblox::MapDerializationAction (C++ type), 117
voxblox::MarchingCubes (C++ class), 85
voxblox::MarchingCubes::~MarchingCubes (C++ function), 86
voxblox::MarchingCubes::calculateVertexConfiguration (C++ function), 86
voxblox::MarchingCubes::interpolateEdgeVertices (C++ function), 86
voxblox::MarchingCubes::interpolateVertex (C++ function), 86
voxblox::MarchingCubes::kEdgeIndexPairs (C++ member), 86
voxblox::MarchingCubes::kTriangleTable (C++ member), 86
voxblox::MarchingCubes::MarchingCubes (C++ function), 86
voxblox::MarchingCubes::meshCube (C++ function), 86
voxblox::MergedTsdfIntegrator (C++ class), 86
voxblox::MergedTsdfIntegrator::bundleRays (C++ function), 87
voxblox::MergedTsdfIntegrator::integratePointCloud (C++ function), 87
voxblox::MergedTsdfIntegrator::integrateRays (C++ function), 87
voxblox::MergedTsdfIntegrator::integrateVoxel (C++ function), 87
voxblox::MergedTsdfIntegrator::integrateVoxels (C++ function), 87
voxblox::MergedTsdfIntegrator::MergedTsdfIntegrator (C++ function), 87
voxblox::mergeLayerAintoLayerB (C++ function), 134
voxblox::mergeVoxelAIntoVoxelB (C++ function), 135
voxblox::Mesh (C++ class), 55
voxblox::Mesh::~Mesh (C++ function), 55
voxblox::Mesh::block_size (C++ member), 56
voxblox::Mesh::clear (C++ function), 55
voxblox::Mesh::clearColors (C++ function), 55
voxblox::Mesh::clearNormals (C++ function), 55
voxblox::Mesh::clearTriangles (C++ function), 55
voxblox::Mesh::colorizeMesh (C++ function), 56
voxblox::Mesh::colors (C++ member), 56
voxblox::Mesh::concatenateMesh (C++ function), 56
voxblox::Mesh::ConstPtr (C++ type), 55
voxblox::Mesh::hasColors (C++ function), 55
voxblox::Mesh::hasNormals (C++ function), 55
voxblox::Mesh::hasTriangles (C++ function), 55
voxblox::Mesh::hasVertices (C++ function), 55
voxblox::Mesh::indices (C++ member), 56
voxblox::Mesh::kInvalidBlockSize (C++ member), 56
voxblox::Mesh::Mesh (C++ function), 55
voxblox::Mesh::normals (C++ member), 56
voxblox::Mesh::origin (C++ member), 56
voxblox::Mesh::Ptr (C++ type), 55
voxblox::Mesh::reserve (C++ function), 55
voxblox::Mesh::resize (C++ function), 55
voxblox::Mesh::size (C++ function), 55
voxblox::Mesh::updated (C++ member), 56
voxblox::Mesh::vertices (C++ member), 56
voxblox::MeshIntegrator (C++ class), 87
voxblox::MeshIntegrator::block_size_ (C++ member), 88
voxblox::MeshIntegrator::block_size_inv_ (C++ member), 88
voxblox::MeshIntegrator::config_ (C++ member), 88
voxblox::MeshIntegrator::cube_index_offsets_ (C++ member), 88
voxblox::MeshIntegrator::extractBlockMesh (C++ function), 88
voxblox::MeshIntegrator::extractMeshInsideBlock (C++ function), 88
voxblox::MeshIntegrator::extractMeshOnBorder (C++ function), 88
voxblox::MeshIntegrator::generateMesh (C++ function), 88
voxblox::MeshIntegrator::generateMeshBlocksFunction (C++ function), 88
voxblox::MeshIntegrator::initFromSdfLayer (C++ function), 87

voxblox::MeshIntegrator::mesh_layer_ (C++ member), 88
voxblox::MeshIntegrator::MeshIntegrator (C++ function), 87, 88
voxblox::MeshIntegrator::sdf_layer_const_ (C++ member), 88
voxblox::MeshIntegrator::sdf_layer_mutable_ (C++ member), 88
voxblox::MeshIntegrator::updateMeshColor (C++ function), 88
voxblox::MeshIntegrator::updateMeshForBlock (C++ function), 88
voxblox::MeshIntegrator::voxel_size_ (C++ member), 88
voxblox::MeshIntegrator::voxel_size_inv_ (C++ member), 88
voxblox::MeshIntegrator::voxels_per_side_ (C++ member), 88
voxblox::MeshIntegrator::voxels_per_side_inv_ (C++ member), 88
voxblox::MeshIntegratorConfig (C++ class), 56
voxblox::MeshIntegratorConfig::integrator_threads (C++ member), 56
voxblox::MeshIntegratorConfig::min_weight (C++ member), 56
voxblox::MeshIntegratorConfig::use_color (C++ member), 56
voxblox::MeshLayer (C++ class), 89
voxblox::MeshLayer::~MeshLayer (C++ function), 89
voxblox::MeshLayer::allocateMeshPtrByCoordinates (C++ function), 89
voxblox::MeshLayer::allocateMeshPtrByIndex (C++ function), 89
voxblox::MeshLayer::allocateNewBlock (C++ function), 89
voxblox::MeshLayer::allocateNewBlockByCoordinates (C++ function), 89
voxblox::MeshLayer::block_size (C++ function), 90
voxblox::MeshLayer::block_size_inv (C++ function), 90
voxblox::MeshLayer::clear (C++ function), 90
voxblox::MeshLayer::clearDistantMesh (C++ function), 89
voxblox::MeshLayer::computeBlockIndexFromCoordinates (C++ function), 89
voxblox::MeshLayer::ConstPtr (C++ type), 89
voxblox::MeshLayer::getAllAllocatedMeshes (C++ function), 89
voxblox::MeshLayer::getAllUpdatedMeshes (C++ function), 89
voxblox::MeshLayer::getMesh (C++ function), 89
voxblox::MeshLayer::getMeshByIndex (C++ function), 89
voxblox::MeshLayer::getMeshPtrByCoordinates (C++ function), 89
voxblox::MeshLayer::getMeshPtrByIndex (C++ function), 89
voxblox::MeshLayer::getNumberOfAllocatedMeshes (C++ function), 90
voxblox::MeshLayer::MeshLayer (C++ function), 89
voxblox::MeshLayer::MeshMap (C++ type), 89
voxblox::MeshLayer::Ptr (C++ type), 89
voxblox::MeshLayer::removeMesh (C++ function), 89
voxblox::MeshLayer::removeMeshByCoordinates (C++ function), 89
voxblox::naiveTransformLayer (C++ function), 135
voxblox::Neighborhood (C++ class), 90
voxblox::Neighborhood::getFromBlockAndVoxelIndex (C++ function), 91
voxblox::Neighborhood::getFromBlockAndVoxelIndexAndDirection (C++ function), 90
voxblox::Neighborhood::getFromGlobalIndex (C++ function), 90
voxblox::Neighborhood::getOffsetBetweenVoxels (C++ function), 91
voxblox::Neighborhood::IndexMatrix (C++ type), 90
voxblox::NeighborhoodLookupTables (C++ class), 91
voxblox::NeighborhoodLookupTables::Distances (C++ type), 91
voxblox::NeighborhoodLookupTables::IndexOffsets (C++ type), 91
voxblox::NeighborhoodLookupTables::kDistances (C++ member), 92
voxblox::NeighborhoodLookupTables::kLongOffsets (C++ member), 92
voxblox::NeighborhoodLookupTables::kOffsets (C++ member), 92
voxblox::NeighborhoodLookupTables::LongIndexOffsets (C++ type), 91
voxblox::normalColorFromNormal (C++ function), 136
voxblox::Object (C++ class), 92
voxblox::Object::~Object (C++ function), 92
voxblox::Object::center_ (C++ member), 93
voxblox::Object::color_ (C++ member), 93
voxblox::Object::getColor (C++ function), 93
voxblox::Object::getDistanceToPoint (C++ function), 92
voxblox::Object::getRayIntersection (C++ function), 93
voxblox::Object::kCube (C++ enumerator), 92
voxblox::Object::kCylinder (C++ enumerator), 92
voxblox::Object::kPlane (C++ enumerator), 92
voxblox::Object::kSphere (C++ enumerator), 92
voxblox::Object::Object (C++ function), 92
voxblox::Object::Type (C++ type), 92
voxblox::Object::type_ (C++ member), 93
voxblox::OccupancyIntegrator (C++ class), 93
voxblox::OccupancyIntegrator::block_size_ (C++ member), 93
voxblox::OccupancyIntegrator::block_size_inv_ (C++ member), 94

voxblox::OccupancyIntegrator::clamp_max_log_ (C++ member), 94
voxblox::OccupancyIntegrator::clamp_min_log_ (C++ member), 94
voxblox::OccupancyIntegrator::Config (C++ class), 57, 94
voxblox::OccupancyIntegrator::Config::max_ray_length_m (C++ member), 57, 94
voxblox::OccupancyIntegrator::Config::min_ray_length_m (C++ member), 57, 94
voxblox::OccupancyIntegrator::Config::probability_hit (C++ member), 57, 94
voxblox::OccupancyIntegrator::Config::probability_miss (C++ member), 57, 94
voxblox::OccupancyIntegrator::Config::threshold_max (C++ member), 57, 94
voxblox::OccupancyIntegrator::Config::threshold_min (C++ member), 57, 94
voxblox::OccupancyIntegrator::Config::threshold_occupancy (C++ member), 57, 94
voxblox::OccupancyIntegrator::config_ (C++ member), 93
voxblox::OccupancyIntegrator::integratePointCloud (C++ function), 93
voxblox::OccupancyIntegrator::layer_ (C++ member), 93
voxblox::OccupancyIntegrator::min_occupancy_log_ (C++ member), 94
voxblox::OccupancyIntegrator::OccupancyIntegrator (C++ function), 93
voxblox::OccupancyIntegrator::prob_hit_log_ (C++ member), 94
voxblox::OccupancyIntegrator::prob_miss_log_ (C++ member), 94
voxblox::OccupancyIntegrator::updateOccupancyVoxel (C++ function), 93
voxblox::OccupancyIntegrator::voxel_size_ (C++ member), 93
voxblox::OccupancyIntegrator::voxel_size_inv_ (C++ member), 94
voxblox::OccupancyIntegrator::voxels_per_side_ (C++ member), 93
voxblox::OccupancyIntegrator::voxels_per_side_inv_ (C++ member), 94
voxblox::OccupancyMap (C++ class), 94
voxblox::OccupancyMap::~OccupancyMap (C++ function), 95
voxblox::OccupancyMap::block_size (C++ function), 95
voxblox::OccupancyMap::block_size_ (C++ member), 95
voxblox::OccupancyMap::Config (C++ class), 57, 95
voxblox::OccupancyMap::Config::occupancy_voxel_size (C++ member), 57, 95
voxblox::OccupancyMap::Config::occupancy_voxels_per_side (C++ member), 57, 95
voxblox::OccupancyMap::getOccupancyLayer (C++ function), 95
voxblox::OccupancyMap::getOccupancyLayerPtr (C++ function), 95
voxblox::OccupancyMap::occupancy_layer_ (C++ member), 95
voxblox::OccupancyMap::OccupancyMap (C++ function), 95
voxblox::OccupancyMap::Ptr (C++ type), 94
voxblox::OccupancyVoxel (C++ class), 57
voxblox::OccupancyVoxel::observed (C++ member), 57
voxblox::OccupancyVoxel::probability_log (C++ member), 57
voxblox::outputMeshAsPly (C++ function), 136
voxblox::outputMeshLayerAsPly (C++ function), 136
voxblox::Plane (C++ class), 95
voxblox::Plane::~Plane (C++ function), 95
voxblox::Plane::distance (C++ function), 96
voxblox::Plane::isPointInside (C++ function), 95
voxblox::Plane::normal (C++ function), 95
voxblox::Plane::Plane (C++ function), 95
voxblox::Plane::setFromDistanceNormal (C++ function), 95
voxblox::Plane::setFromPoints (C++ function), 95
voxblox::PlaneObject (C++ class), 96
voxblox::PlaneObject::getDistanceToPoint (C++ function), 96
voxblox::PlaneObject::getRayIntersection (C++ function), 96
voxblox::PlaneObject::normal_ (C++ member), 96
voxblox::PlaneObject::PlaneObject (C++ function), 96
voxblox::Point (C++ type), 157
voxblox::Pointcloud (C++ type), 157
voxblox::pointcloudToPclXYZ (C++ function), 137
voxblox::pointcloudToPclXYZI (C++ function), 137
voxblox::pointcloudToPclXYZRGB (C++ function), 137
voxblox::PointsMatrix (C++ type), 157
voxblox::probabilityFromLogOdds (C++ function), 137
voxblox::Quaternion (C++ type), 157
voxblox::RainbowColorMap (C++ class), 97
voxblox::rainbowColorMap (C++ function), 137
voxblox::RainbowColorMap::colorLookup (C++ function), 97
voxblox::randomColor (C++ function), 137
voxblox::Ray (C++ type), 157
voxblox::RayCaster (C++ class), 97
voxblox::RayCaster::nextRayIndex (C++ function), 97
voxblox::RayCaster::RayCaster (C++ function), 97
voxblox::recolorVoxbloxMeshMsgByIntensity (C++ function), 138
voxblox::resampleLayer (C++ function), 138
voxblox::Rotation (C++ type), 157
voxblox::serializeLayerAsMsg (C++ function), 138
voxblox::SignedIndex (C++ type), 158

voxblox::signum (C++ function), 138
voxblox::SimpleTsdfIntegrator (C++ class), 97
voxblox::SimpleTsdfIntegrator::integrateFunction (C++ function), 98
voxblox::SimpleTsdfIntegrator::integratePointCloud (C++ function), 98
voxblox::SimpleTsdfIntegrator::SimpleTsdfIntegrator (C++ function), 98
voxblox::SimulationServer (C++ class), 98
voxblox::SimulationServer::~SimulationServer (C++ function), 98
voxblox::SimulationServer::add_robot_pose_ (C++ member), 99
voxblox::SimulationServer::depth_camera_resolution_ (C++ member), 99
voxblox::SimulationServer::esdf_gt_ (C++ member), 99
voxblox::SimulationServer::esdf_gt_pub_ (C++ member), 99
voxblox::SimulationServer::esdf_integrator_ (C++ member), 100
voxblox::SimulationServer::esdf_max_distance_ (C++ member), 99
voxblox::SimulationServer::esdf_occ_integrator_ (C++ member), 100
voxblox::SimulationServer::esdf_test_ (C++ member), 100
voxblox::SimulationServer::esdf_test_pub_ (C++ member), 99
voxblox::SimulationServer::evaluate (C++ function), 98
voxblox::SimulationServer::fov_h_rad_ (C++ member), 99
voxblox::SimulationServer::generate_mesh_ (C++ member), 99
voxblox::SimulationServer::generate_occupancy_ (C++ member), 99
voxblox::SimulationServer::generatePlausibleViewpoint (C++ function), 99
voxblox::SimulationServer::generateSDF (C++ function), 98
voxblox::SimulationServer::getServerConfigFromRosParam (C++ function), 99
voxblox::SimulationServer::incremental_ (C++ member), 99
voxblox::SimulationServer::max_attempts_to_generate_viewpoint_ (C++ member), 99
voxblox::SimulationServer::max_dist_ (C++ member), 99
voxblox::SimulationServer::min_dist_ (C++ member), 99
voxblox::SimulationServer::nh_ (C++ member), 99
voxblox::SimulationServer::nh_private_ (C++ member), 99
voxblox::SimulationServer::num_viewpoints_ (C++ member), 99
voxblox::SimulationServer::occ_integrator_ (C++ member), 100
voxblox::SimulationServer::occ_test_ (C++ member), 100
voxblox::SimulationServer::prepareWorld (C++ function), 98
voxblox::SimulationServer::run (C++ function), 98
voxblox::SimulationServer::sim_pub_ (C++ member), 99
voxblox::SimulationServer::SimulationServer (C++ function), 98
voxblox::SimulationServer::truncation_distance_ (C++ member), 99
voxblox::SimulationServer::tsdf_gt_ (C++ member), 99
voxblox::SimulationServer::tsdf_gt_mesh_pub_ (C++ member), 99
voxblox::SimulationServer::tsdf_gt_pub_ (C++ member), 99
voxblox::SimulationServer::tsdf_integrator_ (C++ member), 100
voxblox::SimulationServer::tsdf_test_ (C++ member), 100
voxblox::SimulationServer::tsdf_test_mesh_pub_ (C++ member), 99
voxblox::SimulationServer::tsdf_test_pub_ (C++ member), 99
voxblox::SimulationServer::view_ptcloud_pub_ (C++ member), 99
voxblox::SimulationServer::visualization_slice_level_ (C++ member), 99
voxblox::SimulationServer::visualize (C++ function), 98
voxblox::SimulationServer::visualize_ (C++ member), 99
voxblox::SimulationServer::voxel_size_ (C++ member), 99
voxblox::SimulationServer::voxels_per_side_ (C++ member), 99
voxblox::SimulationServer::world_ (C++ member), 99
voxblox::SimulationServer::world_frame_ (C++ member), 99
voxblox::SimulationWorld (C++ class), 100
voxblox::SimulationWorld::~SimulationWorld (C++ function), 100
voxblox::SimulationWorld::addGroundLevel (C++ function), 100
voxblox::SimulationWorld::addObject (C++ function), 100
voxblox::SimulationWorld::addPlaneBoundaries (C++ function), 100
voxblox::SimulationWorld::clear (C++ function), 100
voxblox::SimulationWorld::generateSdfFromWorld (C++ function), 101
voxblox::SimulationWorld::getDistanceToPoint (C++ function), 101
voxblox::SimulationWorld::getMaxBound (C++ function), 101
voxblox::SimulationWorld::getMinBound (C++ func-

tion), 101
voxblox::SimulationWorld::getNoisyPointcloudFromTransform (C++ function), 101
voxblox::SimulationWorld::getNoisyPointcloudFromViewpoint (C++ function), 101
voxblox::SimulationWorld::getPointcloudFromTransform (C++ function), 100
voxblox::SimulationWorld::getPointcloudFromViewpoint (C++ function), 100
voxblox::SimulationWorld::setBounds (C++ function), 101
voxblox::SimulationWorld::setVoxel (C++ function), 101
voxblox::SimulationWorld::SimulationWorld (C++ function), 100
voxblox::Sphere (C++ class), 101
voxblox::Sphere::getDistanceToPoint (C++ function), 102
voxblox::Sphere::getRayIntersection (C++ function), 102
voxblox::Sphere::radius_ (C++ member), 102
voxblox::Sphere::Sphere (C++ function), 102
voxblox::test::fillVoxelWithTestData (C++ function), 138, 139
voxblox::test::LayerTest (C++ class), 102
voxblox::test::LayerTest::CompareBlocks (C++ function), 102
voxblox::test::LayerTest::CompareLayers (C++ function), 102
voxblox::test::LayerTest::CompareVoxel (C++ function), 102
voxblox::test::LayerTest::kTolerance (C++ member), 102
voxblox::test::SetUpTestLayer (C++ function), 139
voxblox::ThreadSafeIndex (C++ class), 103
voxblox::ThreadSafeIndex::getNextIndex (C++ function), 103
voxblox::ThreadSafeIndex::reset (C++ function), 103
voxblox::ThreadSafeIndex::ThreadSafeIndex (C++ function), 103
voxblox::timing::Accumulator (C++ class), 103
voxblox::timing::Accumulator::Accumulator (C++ function), 103
voxblox::timing::Accumulator::Add (C++ function), 103
voxblox::timing::Accumulator::LazyVariance (C++ function), 103
voxblox::timing::Accumulator::Max (C++ function), 103
voxblox::timing::Accumulator::Mean (C++ function), 103
voxblox::timing::Accumulator::Min (C++ function), 103
voxblox::timing::Accumulator::RollingMean (C++ function), 103
voxblox::timing::Accumulator::Sum (C++ function), 103
voxblox::timing::Accumulator::TotalSamples (C++ function), 103
voxblox::timing::DebugTimer (C++ type), 158
voxblox::timing::DummyTimer (C++ class), 104
voxblox::timing::DummyTimer::~DummyTimer (C++ function), 104
voxblox::timing::DummyTimer::DummyTimer (C++ function), 104
voxblox::timing::DummyTimer::IsTiming (C++ function), 104
voxblox::timing::DummyTimer::Start (C++ function), 104
voxblox::timing::DummyTimer::Stop (C++ function), 104
voxblox::timing::Timer (C++ class), 104
voxblox::timing::Timer::~Timer (C++ function), 104
voxblox::timing::Timer::IsTiming (C++ function), 104
voxblox::timing::Timer::Start (C++ function), 104
voxblox::timing::Timer::Stop (C++ function), 104
voxblox::timing::Timer::Timer (C++ function), 104
voxblox::timing::TimerMapValue (C++ class), 58
voxblox::timing::TimerMapValue::acc_ (C++ member), 58
voxblox::timing::TimerMapValue::TimerMapValue (C++ function), 58
voxblox::timing::Timing (C++ class), 105
voxblox::timing::Timing::GetHandle (C++ function), 105
voxblox::timing::Timing::GetHz (C++ function), 105
voxblox::timing::Timing::GetMaxSeconds (C++ function), 105
voxblox::timing::Timing::GetMeanSeconds (C++ function), 105
voxblox::timing::Timing::GetMinSeconds (C++ function), 105
voxblox::timing::Timing::GetNumSamples (C++ function), 105
voxblox::timing::Timing::GetTag (C++ function), 105
voxblox::timing::Timing::GetTimers (C++ function), 105
voxblox::timing::Timing::GetTotalSeconds (C++ function), 105
voxblox::timing::Timing::GetVarianceSeconds (C++ function), 105
voxblox::timing::Timing::map_t (C++ type), 105
voxblox::timing::Timing::Print (C++ function), 105
voxblox::timing::Timing::Reset (C++ function), 105
voxblox::timing::Timing::SecondsToTimeString (C++ function), 105
voxblox::toConnectedPCLPolygonMesh (C++ function), 140
voxblox::toSimplifiedPCLPolygonMesh (C++ function), 140
voxblox::Transformation (C++ type), 159
voxblox::Transformer (C++ class), 106
voxblox::Transformer::lookupTransform (C++ function), 106
voxblox::Transformer::transformCallback (C++ function), 106
voxblox::Transformer::Transformer (C++ function), 106

voxblox::transformLayer (C++ function), 140
voxblox::transformPointcloud (C++ function), 141
voxblox::Triangle (C++ type), 159
voxblox::TriangleVector (C++ type), 159
voxblox::TsdfIntegratorBase (C++ class), 106
voxblox::TsdfIntegratorBase::allocateStorageAndGetVoxelPtr (C++ function), 107
voxblox::TsdfIntegratorBase::block_size_ (C++ member), 108
voxblox::TsdfIntegratorBase::block_size_inv_ (C++ member), 108
voxblox::TsdfIntegratorBase::computeDistance (C++ function), 107
voxblox::TsdfIntegratorBase::Config (C++ class), 58, 108
voxblox::TsdfIntegratorBase::Config::allow_clear (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::clear_checks_every_n_frames (C++ member), 59, 109
voxblox::TsdfIntegratorBase::Config::default_truncation_distance (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::enable_anti_grazing (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::integrator_threads (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::max_consecutive_ray_collisions (C++ member), 59, 108
voxblox::TsdfIntegratorBase::Config::max_integration_time_s (C++ member), 59, 109
voxblox::TsdfIntegratorBase::Config::max_ray_length_m (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::max_weight (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::min_ray_length_m (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::sparsity_compensation_factor (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::start_voxel_subsampling_factor (C++ member), 59, 108
voxblox::TsdfIntegratorBase::Config::use_const_weight (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::use_sparsity_compensation_factor (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::use_weight_dropoff (C++ member), 58, 108
voxblox::TsdfIntegratorBase::Config::voxel_carving_enabled (C++ member), 58, 108
voxblox::TsdfIntegratorBase::config_ (C++ member), 108
voxblox::TsdfIntegratorBase::getConfig (C++ function), 107
voxblox::TsdfIntegratorBase::getVoxelWeight (C++ function), 107
voxblox::TsdfIntegratorBase::integratePointCloud (C++ function), 107
voxblox::TsdfIntegratorBase::isPointValid (C++ function), 107
voxblox::TsdfIntegratorBase::layer_ (C++ member), 108
voxblox::TsdfIntegratorBase::mutexes_ (C++ member), 108
voxblox::TsdfIntegratorBase::Ptr (C++ type), 107
voxblox::TsdfIntegratorBase::temp_block_map_ (C++ member), 108
voxblox::TsdfIntegratorBase::temp_block_mutex_ (C++ member), 108
voxblox::TsdfIntegratorBase::TsdfIntegratorBase (C++ function), 107
voxblox::TsdfIntegratorBase::updateLayerWithStoredBlocks (C++ function), 107
voxblox::TsdfIntegratorBase::updateTsdfVoxel (C++ function), 107
voxblox::TsdfIntegratorBase::voxel_size_ (C++ member), 108
voxblox::TsdfIntegratorBase::voxel_size_inv_ (C++ member), 108
voxblox::TsdfIntegratorBase::voxels_per_side_ (C++ member), 108
voxblox::TsdfIntegratorBase::voxels_per_side_inv_ (C++ member), 108
voxblox::TsdfIntegratorFactory (C++ class), 109
voxblox::TsdfIntegratorFactory::create (C++ function), 109
voxblox::TsdfIntegratorType (C++ type), 117
voxblox::TsdfMap (C++ class), 109
voxblox::TsdfMap::~TsdfMap (C++ function), 110
voxblox::TsdfMap::block_size (C++ function), 110
voxblox::TsdfMap::block_size_ (C++ member), 110
voxblox::TsdfMap::Config (C++ class), 59, 110
voxblox::TsdfMap::Config::tsdf_voxel_size (C++ member), 59, 110
voxblox::TsdfMap::Config::tsdf_voxels_per_side (C++ member), 59, 110
voxblox::TsdfMap::coordPlaneSliceGetDistanceWeight (C++ function), 110
voxblox::TsdfMap::EigenDRef (C++ type), 109
voxblox::TsdfMap::EigenDStride (C++ type), 109
voxblox::TsdfMap::getTsdfLayer (C++ function), 110
voxblox::TsdfMap::getTsdfLayerPtr (C++ function), 110
voxblox::TsdfMap::Ptr (C++ type), 109
voxblox::TsdfMap::tsdf_layer_ (C++ member), 110
voxblox::TsdfMap::TsdfMap (C++ function), 110
voxblox::TsdfMap::voxel_size (C++ function), 110
voxblox::TsdfServer (C++ class), 111
voxblox::TsdfServer::~TsdfServer (C++ function), 111
voxblox::TsdfServer::accumulate_icp_corrections_ (C++ member), 113
voxblox::TsdfServer::cache_mesh_ (C++ member), 113
voxblox::TsdfServer::cached_mesh_msg_ (C++ member), 114

voxblox::TsdfServer::clear (C++ function), 112
voxblox::TsdfServer::clear_map_srv_ (C++ member), 114
voxblox::TsdfServer::clearMapCallback (C++ function), 112
voxblox::TsdfServer::color_map_ (C++ member), 113
voxblox::TsdfServer::color_mode_ (C++ member), 113
voxblox::TsdfServer::enable_icp_ (C++ member), 113
voxblox::TsdfServer::freespace_pointcloud_queue_ (C++ member), 114
voxblox::TsdfServer::freespace_pointcloud_sub_ (C++ member), 113
voxblox::TsdfServer::generate_mesh_srv_ (C++ member), 114
voxblox::TsdfServer::generateMesh (C++ function), 111
voxblox::TsdfServer::generateMeshCallback (C++ function), 112
voxblox::TsdfServer::getNextPointcloudFromQueue (C++ function), 112
voxblox::TsdfServer::getServerConfigFromRosParam (C++ function), 111
voxblox::TsdfServer::getSliceLevel (C++ function), 112
voxblox::TsdfServer::getTsdfMapPtr (C++ function), 112
voxblox::TsdfServer::getWorldFrame (C++ function), 112
voxblox::TsdfServer::icp_ (C++ member), 114
voxblox::TsdfServer::icp_corrected_frame_ (C++ member), 113
voxblox::TsdfServer::icp_corrected_transform_ (C++ member), 114
voxblox::TsdfServer::icp_transform_pub_ (C++ member), 114
voxblox::TsdfServer::insertFreespacePointcloud (C++ function), 111
voxblox::TsdfServer::insertPointcloud (C++ function), 111
voxblox::TsdfServer::integratePointcloud (C++ function), 111
voxblox::TsdfServer::last_msg_time_freespace_ptcloud_ (C++ member), 114
voxblox::TsdfServer::last_msg_time_ptcloud_ (C++ member), 114
voxblox::TsdfServer::load_map_srv_ (C++ member), 114
voxblox::TsdfServer::loadMap (C++ function), 112
voxblox::TsdfServer::loadMapCallback (C++ function), 112
voxblox::TsdfServer::max_block_distance_from_body_ (C++ member), 113
voxblox::TsdfServer::mesh_filename_ (C++ member), 113
voxblox::TsdfServer::mesh_integrator_ (C++ member), 114
voxblox::TsdfServer::mesh_layer_ (C++ member), 114
voxblox::TsdfServer::mesh_pub_ (C++ member), 113
voxblox::TsdfServer::min_time_between_msgs_ (C++ member), 113
voxblox::TsdfServer::newPoseCallback (C++ function), 111
voxblox::TsdfServer::nh_ (C++ member), 112
voxblox::TsdfServer::nh_private_ (C++ member), 112
voxblox::TsdfServer::occupancy_marker_pub_ (C++ member), 114
voxblox::TsdfServer::pointcloud_queue_ (C++ member), 114
voxblox::TsdfServer::pointcloud_queue_size_ (C++ member), 113
voxblox::TsdfServer::pointcloud_sub_ (C++ member), 113
voxblox::TsdfServer::pose_corrected_frame_ (C++ member), 113
voxblox::TsdfServer::processPointCloudMessageAndInsert (C++ function), 111
voxblox::TsdfServer::publish_pointclouds_ (C++ member), 113
voxblox::TsdfServer::publish_pointclouds_srv_ (C++ member), 114
voxblox::TsdfServer::publish_slices_ (C++ member), 113
voxblox::TsdfServer::publish_tsdf_info_ (C++ member), 113
voxblox::TsdfServer::publish_tsdf_map_ (C++ member), 113
voxblox::TsdfServer::publish_tsdf_map_srv_ (C++ member), 114
voxblox::TsdfServer::publishAllUpdatedTsdfVoxels (C++ function), 111
voxblox::TsdfServer::publishMap (C++ function), 111
voxblox::TsdfServer::publishPointclouds (C++ function), 111
voxblox::TsdfServer::publishPointcloudsCallback (C++ function), 112
voxblox::TsdfServer::publishSlices (C++ function), 111
voxblox::TsdfServer::publishTsdfMapCallback (C++ function), 112
voxblox::TsdfServer::publishTsdfOccupiedNodes (C++ function), 111
voxblox::TsdfServer::publishTsdfSurfacePoints (C++ function), 111
voxblox::TsdfServer::save_map_srv_ (C++ member), 114
voxblox::TsdfServer::saveMap (C++ function), 111
voxblox::TsdfServer::saveMapCallback (C++ function), 112
voxblox::TsdfServer::setPublishSlices (C++ function), 112
voxblox::TsdfServer::setSliceLevel (C++ function), 112
voxblox::TsdfServer::setWorldFrame (C++ function),

112
voxblox::TsdfServer::slice_level_ (C++ member), 113
voxblox::TsdfServer::surface_pointcloud_pub_ (C++ member), 114
voxblox::TsdfServer::tf_broadcaster_ (C++ member), 114
voxblox::TsdfServer::transformer_ (C++ member), 114
voxblox::TsdfServer::tsdf_integrator_ (C++ member), 114
voxblox::TsdfServer::tsdf_map_ (C++ member), 114
voxblox::TsdfServer::tsdf_map_pub_ (C++ member), 114
voxblox::TsdfServer::tsdf_map_sub_ (C++ member), 114
voxblox::TsdfServer::tsdf_pointcloud_pub_ (C++ member), 114
voxblox::TsdfServer::tsdf_slice_pub_ (C++ member), 114
voxblox::TsdfServer::tsdfMapCallback (C++ function), 112
voxblox::TsdfServer::TsdfServer (C++ function), 111
voxblox::TsdfServer::update_mesh_timer_ (C++ member), 114
voxblox::TsdfServer::updateMesh (C++ function), 111
voxblox::TsdfServer::updateMeshEvent (C++ function), 112
voxblox::TsdfServer::use_freespace_pointcloud_ (C++ member), 113
voxblox::TsdfServer::verbose_ (C++ member), 112
voxblox::TsdfServer::world_frame_ (C++ member), 112
voxblox::TsdfVoxel (C++ class), 59
voxblox::TsdfVoxel::color (C++ member), 59
voxblox::TsdfVoxel::distance (C++ member), 59
voxblox::TsdfVoxel::weight (C++ member), 59
voxblox::utils::centerBlocksOfLayer (C++ function), 141
voxblox::utils::clearSphereAroundPoint (C++ function), 141
voxblox::utils::computeMapBoundsFromLayer (C++ function), 141
voxblox::utils::computeVoxelError (C++ function), 141
voxblox::utils::evaluateLayersRmse (C++ function), 142
voxblox::utils::fillSphereAroundPoint (C++ function), 142
voxblox::utils::getAndAllocateSphereAroundPoint (C++ function), 142
voxblox::utils::getColorIfValid (C++ function), 143
voxblox::utils::getSdfIfValid (C++ function), 143, 144
voxblox::utils::getSphereAroundPoint (C++ function), 144
voxblox::utils::getVoxelSdf (C++ function), 144
voxblox::utils::isObservedVoxel (C++ function), 144, 145
voxblox::utils::isSameBlock (C++ function), 145
voxblox::utils::isSameLayer (C++ function), 145
voxblox::utils::isSameVoxel (C++ function), 145, 146
voxblox::utils::kEvaluateAllVoxels (C++ enumerator), 117
voxblox::utils::kEvaluated (C++ enumerator), 117
voxblox::utils::kIgnored (C++ enumerator), 117
voxblox::utils::kIgnoreErrorBehindAllSurfaces (C++ enumerator), 117
voxblox::utils::kIgnoreErrorBehindGtSurface (C++ enumerator), 117
voxblox::utils::kIgnoreErrorBehindTestSurface (C++ enumerator), 117
voxblox::utils::kNoOverlap (C++ enumerator), 117
voxblox::utils::readProtoMsgCountToStream (C++ function), 146
voxblox::utils::readProtoMsgFromStream (C++ function), 146
voxblox::utils::setVoxelSdf (C++ function), 146, 147
voxblox::utils::setVoxelWeight (C++ function), 147
voxblox::utils::VoxelEvaluationDetails (C++ class), 60
voxblox::utils::VoxelEvaluationDetails::max_error (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::min_error (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::num_evaluated_voxels (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::num_ignored_voxels (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::num_non_overlapping_voxels (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::num_overlapping_voxels (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::rmse (C++ member), 60
voxblox::utils::VoxelEvaluationDetails::toString (C++ function), 60
voxblox::utils::VoxelEvaluationMode (C++ type), 117
voxblox::utils::VoxelEvaluationResult (C++ type), 117
voxblox::utils::writeProtoMsgCountToStream (C++ function), 147
voxblox::utils::writeProtoMsgToStream (C++ function), 148
voxblox::VertexIndex (C++ type), 159
voxblox::VertexIndexList (C++ type), 159
voxblox::visualizeDistanceIntensityEsdfVoxels (C++ function), 148
voxblox::visualizeDistanceIntensityEsdfVoxelsSlice (C++ function), 148
voxblox::visualizeDistanceIntensityTsdfVoxels (C++ function), 148
voxblox::visualizeDistanceIntensityTsdfVoxelsNearSurface (C++ function), 149
voxblox::visualizeDistanceIntensityTsdfVoxelsSlice (C++ function), 149
voxblox::visualizeFreeEsdfVoxels (C++ function), 149
voxblox::visualizeIntensityVoxels (C++ function), 149
voxblox::visualizeNearSurfaceTsdfVoxels (C++ func-

tion), 150
voxblox::visualizeOccupiedOccupancyVoxels (C++ function), 150
voxblox::visualizeOccupiedTsdfVoxels (C++ function), 150
voxblox::visualizeTsdfVoxels (C++ function), 150
voxblox::voxel_types::kEsdf (C++ member), 151
voxblox::voxel_types::kIntensity (C++ member), 151
voxblox::voxel_types::kNotSerializable (C++ member), 152
voxblox::voxel_types::kOccupancy (C++ member), 152
voxblox::voxel_types::kTsdf (C++ member), 152
voxblox::VoxelIndex (C++ type), 159
voxblox::VoxelIndexList (C++ type), 159
voxblox::VoxelKey (C++ type), 160
voxblox_rviz_plugin::VoxbloxMeshDisplay (C++ class), 115
voxblox_rviz_plugin::VoxbloxMeshDisplay::~VoxbloxMeshDisplay (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshDisplay::onInitialize (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshDisplay::reset (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshDisplay::VoxbloxMeshDisplay (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshVisual (C++ class), 115
voxblox_rviz_plugin::VoxbloxMeshVisual::~VoxbloxMeshVisual (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshVisual::setFrameOrientation (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshVisual::setFramePosition (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshVisual::setMessage (C++ function), 115
voxblox_rviz_plugin::VoxbloxMeshVisual::VoxbloxMeshVisual (C++ function), 115