Stereo matching and PCL

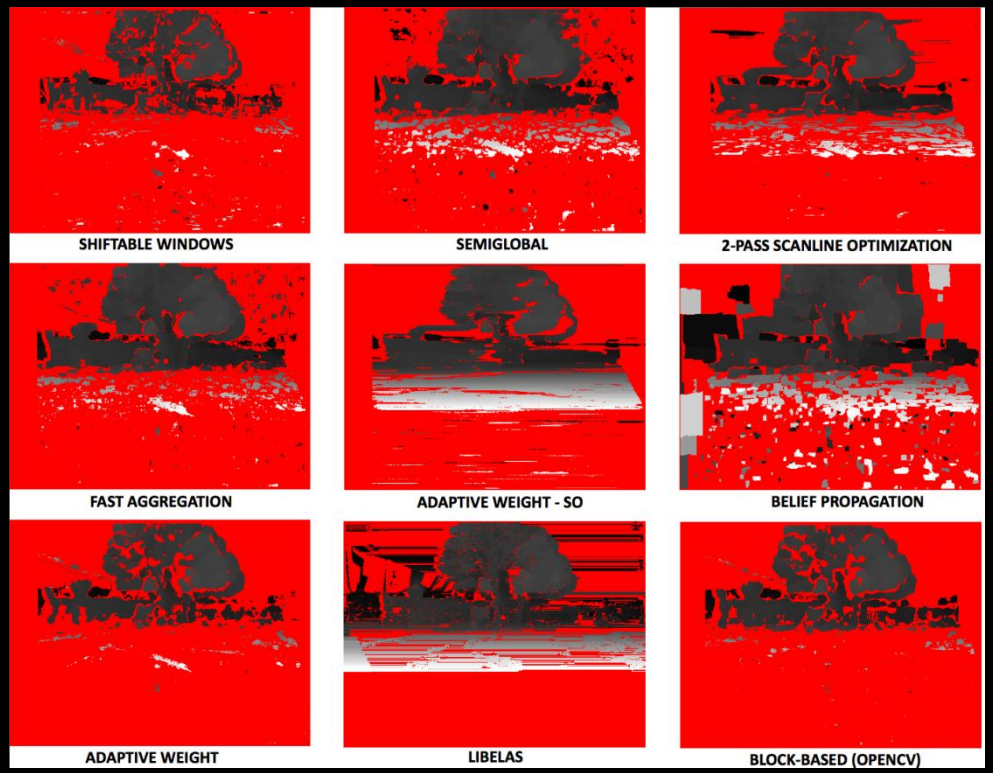
Federico Tombari
CGLibs
Pisa, June 4, 2013

Stereo Overview

- PCL now includes a stereo library for computing point clouds out of rectified stereo image pairs
- Currently, two stereo matching algorithms implemented plus tools for converting the disparity map into a point cloud
- Choice of stereo algorithms targeted for outdoor use, since developed as part of Honda Research Code Sprint (PCL HRCS)

Evaluated Approaches

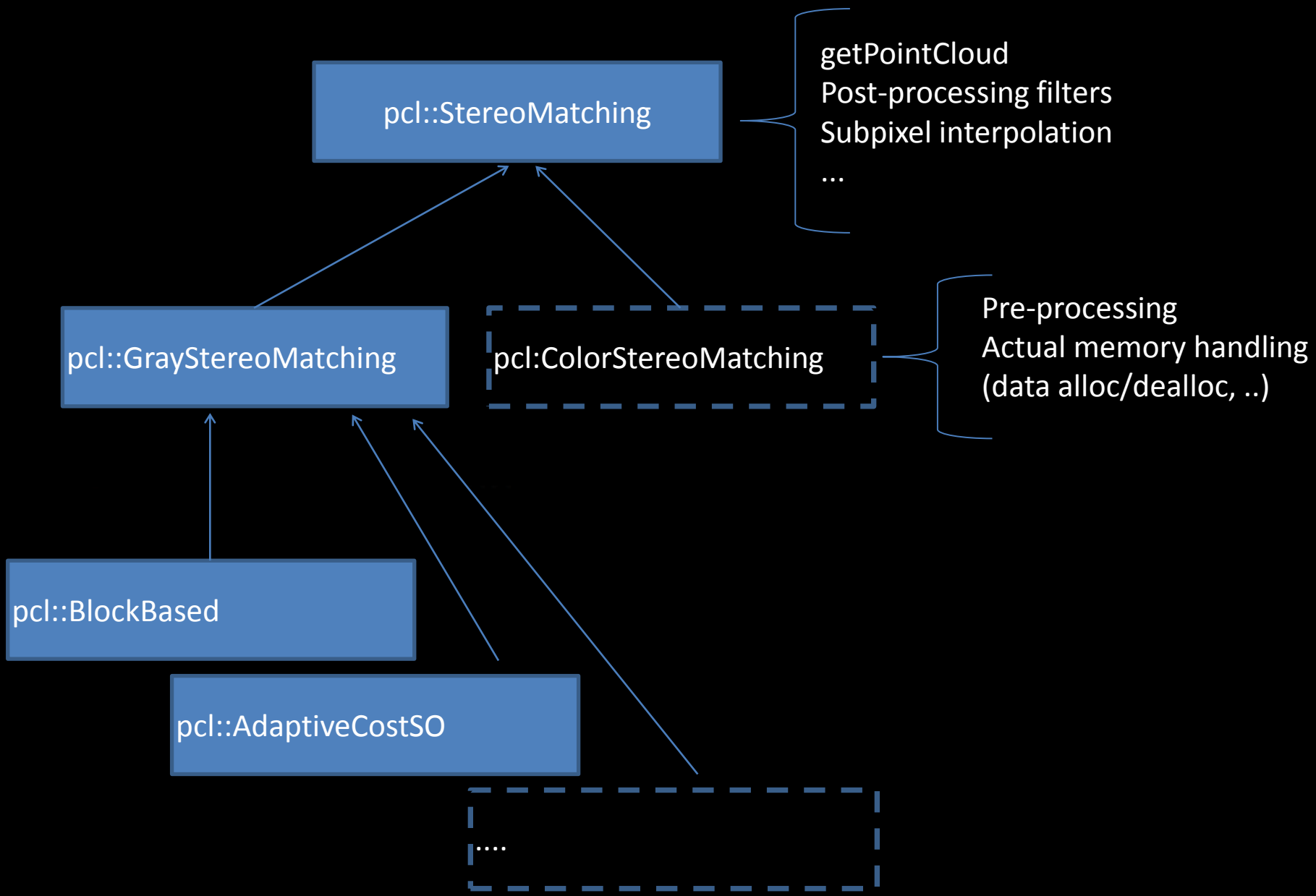
- Several stereo algorithms were evaluated for use in outdoor segmentation tasks



- PCL contains implementations for
  - Block-based
  - Adaptive Weight - SO algorithms (inspired by: Liang et al., “High-quality real-time stereo using adaptive cost aggregation and dynamic programming”, 3DPVT 06
- Efficient implementation via fast incremental schemes (running average)

pointcloudlibrary

Class structure



Stereo Matching

- Input:
  - Rectified Image Pair
  - Camera Calibration Parameters
    - Intrinsic parameters (focal length, principal point coords.)
    - Baseline
- Output:
  - Point Cloud
  - Disparity Image

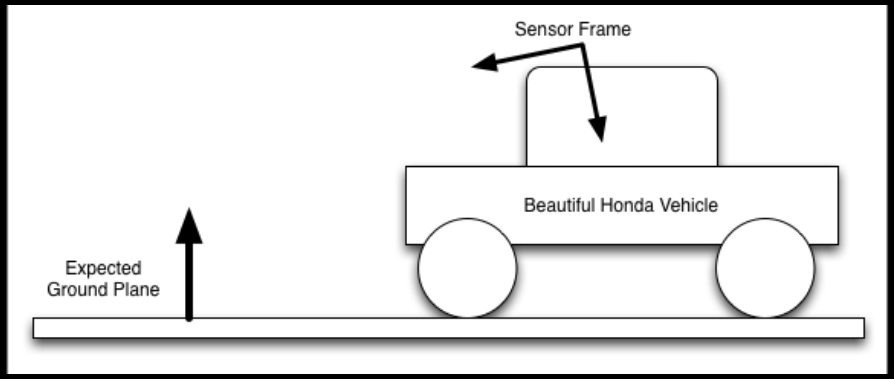
pcl_png2pcd

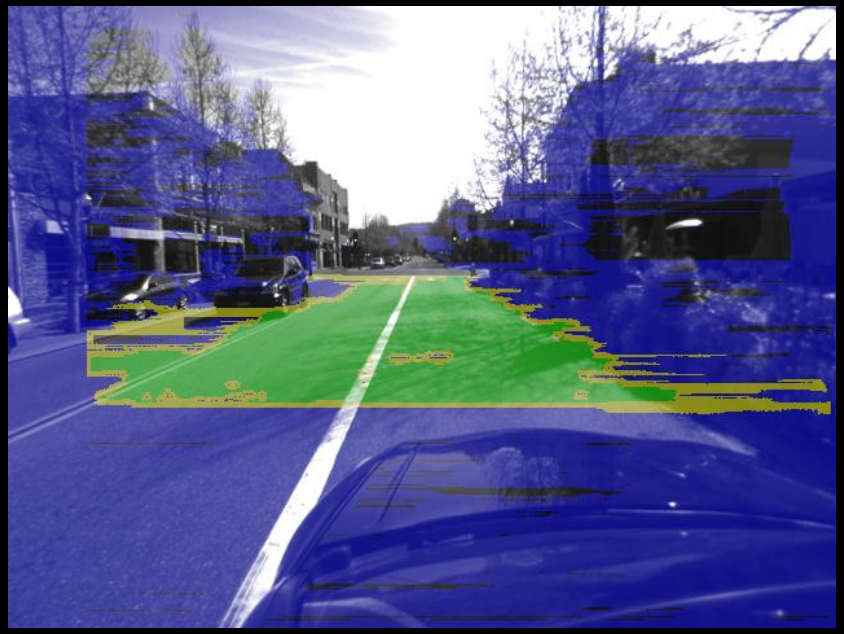
- Command-line tool for png conversion
- Enables easy conversion from images to PCD format, so image data can be treated in the same way as other points
- Developed by Gioia Ballin as part of her Google Summer of Code (GSOC)

Ground Segmentation from Stereo

- Uses connected component segmentation (see related work by Alex Trevor)
- Ground finding, not lane finding
- New comparator was implemented for this purpose

- pcl_stereo_ground_segmentation demo demonstrates this





```
// Load Images (PCD Format)
pcl::PointCloud<pcl::RGB>::Ptr left_cloud (new pcl::PointCloud<pcl::RGB>);
pcl::PointCloud<pcl::RGB>::Ptr right_cloud (new pcl::PointCloud<pcl::RGB>);
pcd.read ("left.pcd", *left_cloud);
pcd.read ("right.pcd", *right_cloud);

// Set up Stereo
pcl::AdaptiveCostSOStereoMatching stereo;
stereo.setMaxDisparity(60);
stereo.setXOffset(0);
stereo.setRadius(5);
stereo.setSmoothWeak(20);
stereo.setSmoothStrong(100);
stereo.setGammaC(25);
stereo.setGammaS(10);
stereo.setRatioFilter(20);
stereo.setPeakFilter(0);
stereo.setLeftRightCheck(true);
stereo.setLeftRightCheckThreshold(1);
stereo.setPreProcessing(true);

// Compute disparities and filter
stereo.compute(*left_cloud,  *right_cloud);
stereo.medianFilter(4);

// Get the output point cloud
pcl::PointCloud<pcl::PointXYZRGB>::Ptr out_cloud( new pcl::PointCloud<pcl::PointXYZRGB> );
// principal point x, principal point y, focal length, baseline, output cloud, input texture
stereo.getPointCloud(318.112200, 224.334900, 368.534700, 0.8387445, out_cloud, left_cloud);
```