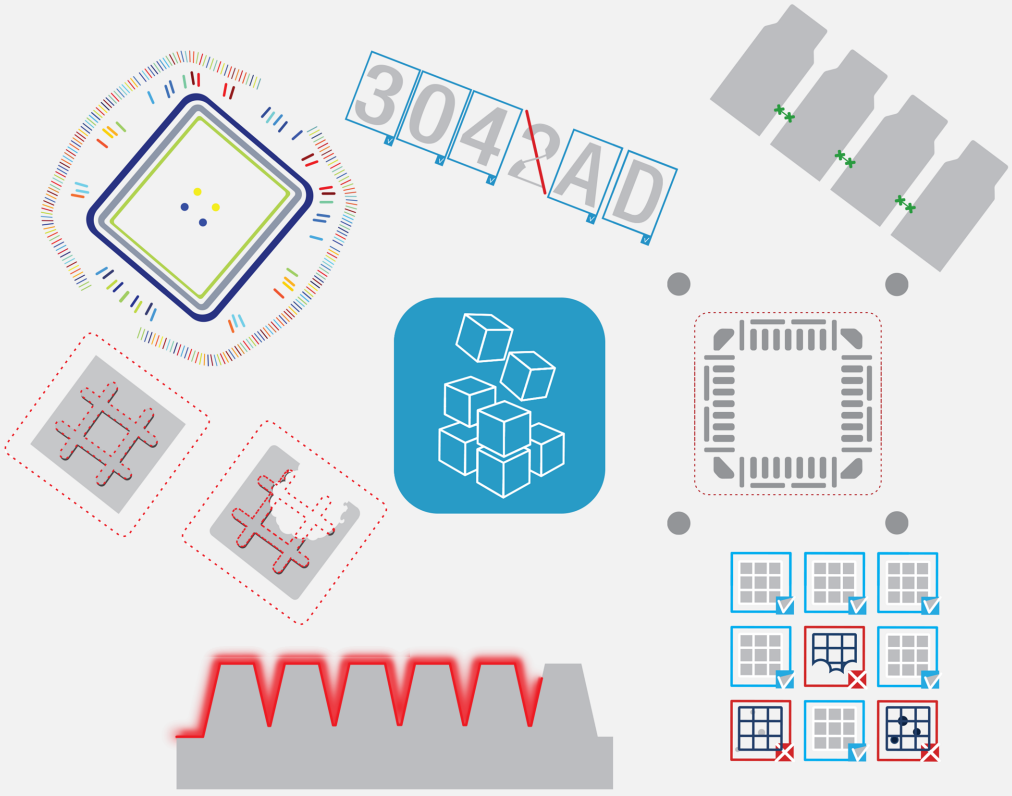
euresys
Empowering Computer Vision

USER GUIDE

Open eVision

Easy3D Compatibility with Zivid 3D Scanners



© EURESYS s.a. 2021 - Document D198EN-Easy3D Compatibility with Zivid 3D Scanners-2.15.1.1151 built on 2021-05-07

This documentation is provided with Open eVision 2.15.1 (doc build 1151).
www.euresys.com

Easy3D Compatibility with Zivid 3D Scanners

Introduction

The **Zivid** 3D sensors are structured-light cameras for industrial application.

The specifications are available on the manufacturer website:
https://www.zivid.com/compare-and-select-3d-cameras

- This document explains how to use the 3D data coming from these sensors with **Open eVision** 3D libraries and tools.
- A sample application distributed with source code demonstrates that integration. This application is freely available in the Easy3D Sensors Compatibility additional resources package on **Euresys** web site.

Resources

This document and the sample applications are based on the following resources:

- **Zivid** One+ Medium (also compatible with all other **Zivid** sensors)
- **Zivid SDK** 2.3.0
- **Open eVision** 2.15
- Microsoft Visual Studio 2017

The **Zivid SDK** is available on the manufacturer website:
https://www.zivid.com/downloads

Features

- The **Zivid SDK** exposes different data types:

Format	Description	Bits per pixel
`PointXYZ`	float32 array of 3D coordinate	96
`PointXYZW`	float32 array of 4D coordinate	128
`PointZ`	float32 Z coordinate	32
`ColorRBGA`	UINT8 RGBA colors	32
`SNR`	float32 Signal-to-Noise ratio values	32
`PointXYZColorRGBA`	float32 array of 3D coordinate and RGBA colors	128
`PointXYZColorBGRA`	float32 array of 3D coordinate and BGRA colors	128

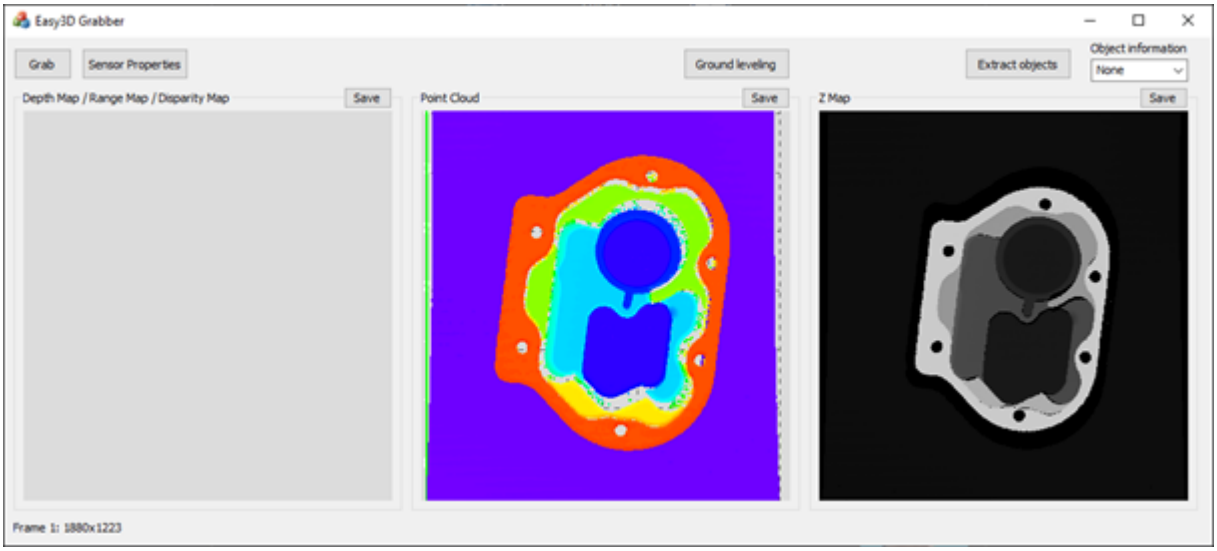
- The XYZ positions in the different data types are expressed in a coordinate system that has its origin on the camera with a Z axis toward the scene.

Easy3DGrab sample application

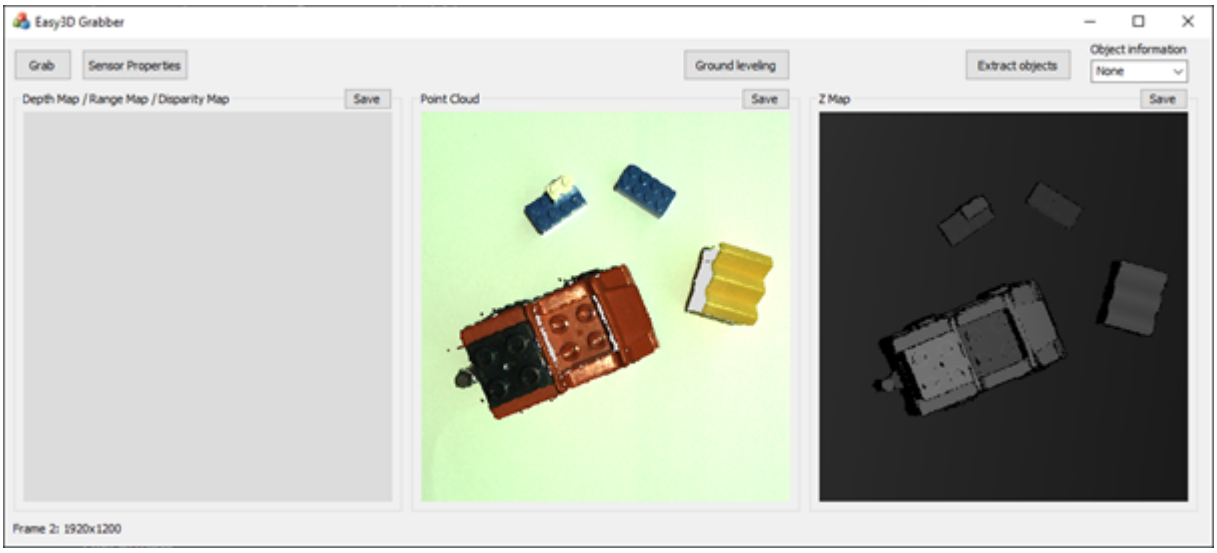
Easy3DGrab is distributed with C++ source code as an **Open eVision** additional resource.

- It features the import of the `PointXYZ` or the `PointXYZColorRGBA` formats and the conversion to **Open eVision** formats (`EPointCloud` and `EZMap`).
- You can save these representations.
- Click on the Grab button to acquire a new image.
- Open the Sensor Properties dialog to:
  - Load a configuration file from **Zivid Studio**.
  - Retrieve or not the color data.
- The Object extraction function is exposed but you can use it only with the Easy3DObject license.
- You can also perform a Ground leveling.

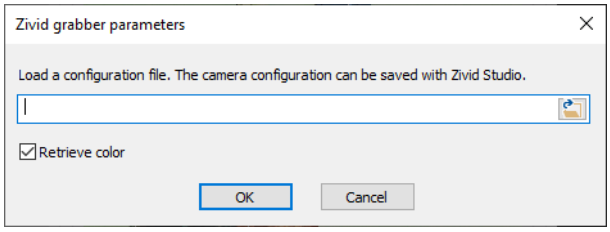
NOTE
When the application starts, the **Zivid Capture Assistant** triggers a capture to determine the sensor parameters.



The Easy3DGrab application:
EDepthMap not available (left), EPointCloud (center), EZMap (right)



The Easy3DGrab application: an EPointCloud (center) retrieved with colors



Setting the 3D sensor parameters

C++ code sample to convert Zivid formats to Easy3D objects

Converting a PointXYZ or a PointXYZRBGA to an EPointCloud

Here is the code snippet to fill an `Easy3D::EPointCloud` object from a Zivid `PointXYZ` or `PointXYZRBGA`:

```
// Configure device
Zivid::Application sensor;
Zivid::Camera camera = sensor.connectCamera();

const auto suggestSettingsParameters =
Zivid::CaptureAssistant::SuggestSettingsParameters{

Zivid::CaptureAssistant::
SuggestSettingsParameters::AmbientLightFrequency::none,
  Zivid::CaptureAssistant::SuggestSettingsParameters::MaxCaptureTime{
  std::chrono::milliseconds{ 1200 } }};
Zivid::Settings settings = Zivid::CaptureAssistant::suggestSettings(camera,
suggestSettingsParameters);

// Capture pointcloud
Zivid::Frame frame = camera.capture(settings);
const auto pointCloudSensor = frame.pointCloud();

// Convert to EPointCloud
Easy3D::EPointCloud pointcloud;
bool retrive_color; // Set to true to retrieve colors

size_t width = pointCloudSensor.width();
size_t height = pointCloudSensor.height();
size_t nbPoints = width * height;

std::vector<Easy3D::E3DPoint> points;
points.reserve(nbPoints);
float max = FLT_MIN;

if (retrive_color)
{
  // Retrieve 3D coordinate and colors
  const auto data = pointCloudSensor.copyData<Zivid::PointXYZColorRGBA>();
  std::vector<EC24A> colors;
  colors.reserve(nbPoints);
  for (size_t i = 0; i < nbPoints; ++i)
  {
    if (!data(i).point.isNaN())
    {
      points.emplace_back(data(i).point.x, -data(i).point.y, data
(i).point.z);
      colors.emplace_back(data(i).color.r, data(i).color.g, data(i).color.b,
                          data(i).color.a);

      if (data(i).point.z > max)
        max = data(i).point.z;
    }
  }
```

```
  // Change Z convention
  for (int i = 0; i < points.size(); ++i)
    points[i].Z = max - points[i].Z;

  pointcloud.AddPoints(points);
  pointcloud.FillAttributeBuffer(Easy3D::E3DAttribute_Color, colors.data());
}
else
{
  // Only retrieve 3D coordinate
  const auto data = pointCloudSensor.copyData<Zivid::PointXYZ>();
  for (size_t i = 0; i < nbPoints; ++i)
  {
    if (!data(i).isNaN())
    {
      points.emplace_back(data(i).x, -data(i).y, data(i).z);

      if (data(i).z > max)
        max = data(i).z;
    }
  }

  // Change Z convention
  for (int i = 0; i < points.size(); ++i)
    points[i].Z = max - points[i].Z;

  pointcloud.AddPoints(points);
}
```

ZMap

- You cannot generate a ZMap (a gray scale image encoding distance from a reference plane, also called an orthographic projection of the point cloud) directly from the **Zivid** 3D sensors.
- Generate a ZMap from the point cloud with the `Easy3D::EPointCloudToZMapConverter` class.

TIP
The sample application **Easy3DGrab** implement these conversions.