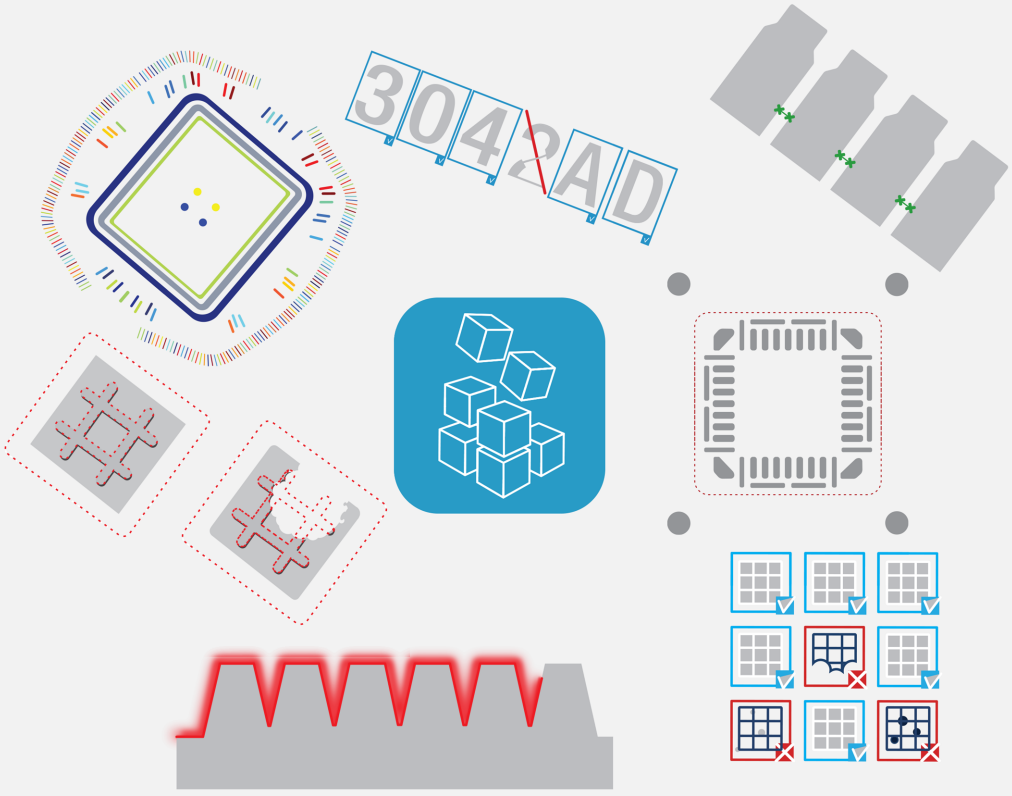
euresys
Empowering Computer Vision

USER GUIDE

Open eVision

Easy3D Compatibility with Photoneo PhoXi 3D Sensors



This documentation is provided with Open eVision 2.15.0 (doc build 1147).
www.euresys.com

Easy3D Compatibility with Photoneo PhoXi 3D Sensors

Introduction

The **Photoneo Phoxi** 3D sensors are structured-light cameras for industrial applications.

The specifications are available on the manufacturer website:
https://www.photoneo.com/phoxi-3d-scanner/

- 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:

- **Photoneo PhoXi 3D Scanner S**
- **Photoneo PhoXi Control SDK** 1.2.22
- **Open eVision** 2.15
- Microsoft Visual Studio 2017

The **Phoxi Control SDK** is available on the manufacturer website:
https://www.photoneo.com/3d-scanning-software/

Features

- The **Phoxi Control SDK** exposes different data types:

Format	Description	Bits per pixel
`PointCloud32f`	float32 array of 3D coordinates	96
`DepthMap32f`	float32 array of depth data	32
`NormalMap32f`	float32 array of normal vectors	96
`Texture32f`	float32 array of grayscale intensities	32
`TextureRGB32f`	float32 array of RGB color textures	96
`ConfidenceMap32f`	float32 array of confidence measurements	32

- The `PointCloud32f` XYZ positions are expressed in a coordinate system centered on the camera with a Z axis towards the scene.

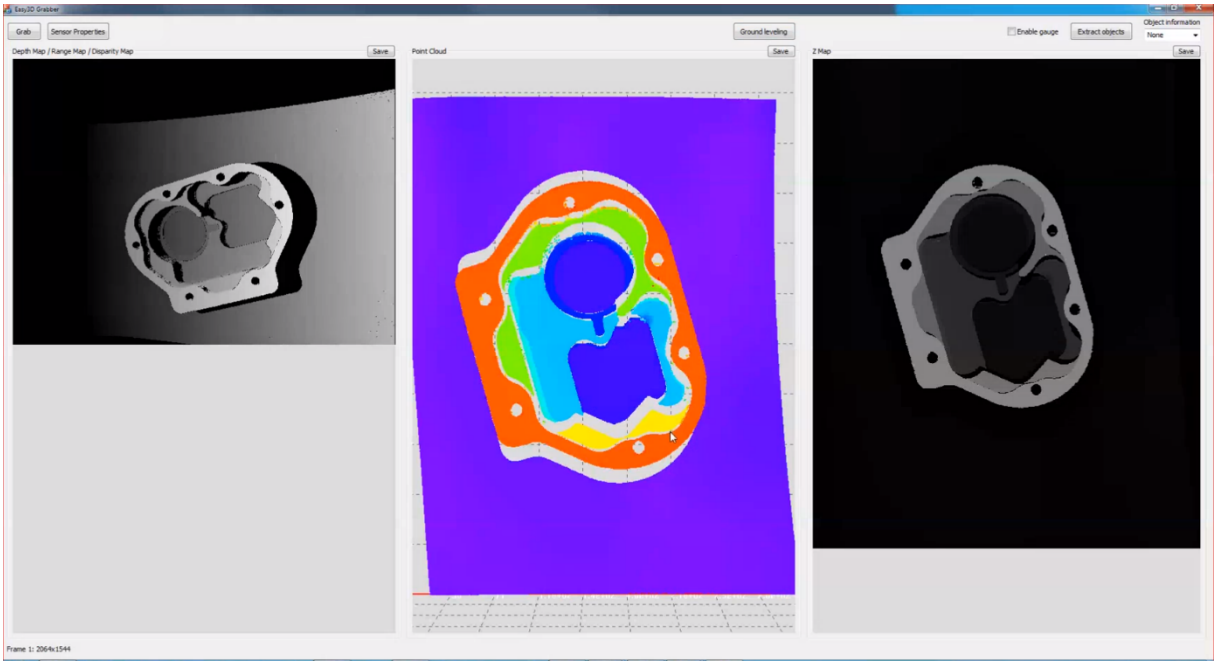
Easy3DGrab sample application

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

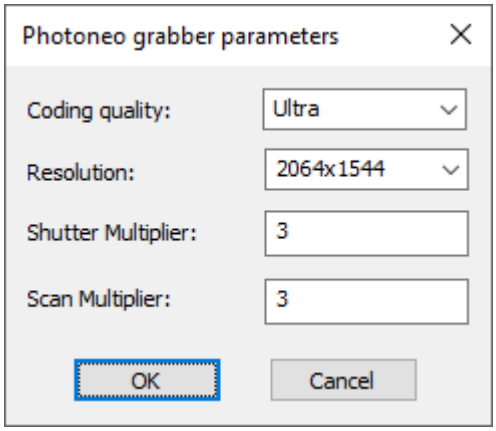
- It features the import of the `PointCloud32f` and the `DepthMap32f` format and the conversion to depth maps, point clouds and ZMaps.
- You can save these representations.
- Click on the Grab button to acquire a new image.
- Open the Sensor Properties dialog to adjust:
  - The *coding quality*, there are 3 options:
    - *Fast*: no sub-pixel accuracy
    - *High*: sub-pixel accuracy (default)
    - *Ultra*: enhanced sub-pixel accuracy
  - The *resolution*: high or low.
  - The *shutter multiplier*: a value in [1, 20].
    - It increases the scanning time by multiplying the projection of patterns. This helps when you scan dark objects, at sharp scanning angles and in any other condition when the pattern is reflected only partially.
    - High values can lead to oversaturation and cause missing points in the point cloud.
  - The *scan multiplier*: a value in [1, 20].
    - It increases the scanning time by repeating and averaging the patterns. This helps to increase the signal-to-noise ratio.
    - It brings a higher contrast when a high dynamic range is required and when the shutter multiplier leads to oversaturation.

- The Object extraction function is exposed but you can use it only with the Easy3DObject license.
- You can also perform a Ground leveling.

> **NOTE**
> You must start **PhoXi Control** before running the **Easy3DGrab** application.



The Easy3DGrab application: EDepthMap (left), EPointCloud (center), EZMap (right)



The 3D sensor parameters

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

Converting a DepthMap32f to an EDepthMap

Here is the code snippet to fill an `Easy3D::EDepthMap16` object from a Photoneo `DepthMap32f`:

```
pho::api::PFrame Frame;
Frame = PhoXiDevice->GetSpecificFrame(FrameID,
pho::api::PhoXiTimeout::Infinity);

if (Frame->DepthMap.Empty())
{
  // Error
}

int width = Frame->DepthMap.Size.Width;
int height = Frame->DepthMap.Size.Height;

// Retrieve the depth map
float *data = (float*)Frame->DepthMap.GetDataPtr();

Easy3D::EDepthMap16 dmap;
dmap.SetSize(width, height);

// Find the minimum and the maximum in the depth map
float max = FLT_MIN, min = FLT_MAX;
int index = 0;
for (int y = 0; y < height; ++y)
{
  for (int x = 0; x < width; ++x, ++index)
  {
    if (data[index] > max)
      max = data[index];

    if (data[index] != 0.0f && data[index] < min)
      min = data[index];
  }
}

float rangeFactor = 65536.0f / (max - min);
index = 0;
for (int y = 0; y < height; ++y)
{
  // Copy each row of the depth map
  uint16_t *dst = (uint16_t*) dmap.GetBufferPtr(0, y);
  for (int x = 0; x < width; ++x, ++index)
  {
    if (data[index] == 0.f)
      dst[x] = 0;
    else
    {
      dst[x] = uint16_t((max - data[index]) * rangeFactor);
    }
  }
}
```

Converting a PointCloud32f to an EPointCloud

Here is the code snippet to fill an `Easy3D::EPointCloud` object from a Photoneo `PointCloud32f`:

```
pho::api::PFrame Frame;
Frame = PhoXiDevice->GetSpecificFrame(FrameID,
pho::api::PhoXiTimeout::Infinity);

if (Frame->PointCloud.Empty())
{
  // Error
}

int width = Frame->PointCloud.Size.Width;
int height = Frame->PointCloud.Size.Height;
int size = width * height;
float max = FLT_MIN;

// Retrieve the point cloud
float *data = (float*)Frame->PointCloud.GetDataPtr();

std::vector<Easy3D::E3DPoint> pts;
pts.reserve(size);

// Change the origin of the point cloud
Easy3D::E3DPoint p;
for (int i = 0; i < size; ++i)
{
  p.X = data[3 * i];
  p.Y = - data[3 * i + 1];
  p.Z = data[3 * i + 2];
  if (p.X != 0.0f && p.Y != 0.0f && p.Z != 0.0f)
  {
    pts.push_back(p);
    if (p.Z > max)
      max = p.Z;
  }
}

for (int i = 0; i < pts.size(); ++i)
  pts[i].Z = max - pts[i].Z;

// Set the point cloud
Easy3D::EPointCloud point_cloud;
point_cloud.AddPoints(pts);
```

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 **Photoneo** 3D sensors.
- Generate a ZMap from the point cloud with the `Easy3D::EPointCloudToZMapConverter` class.

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