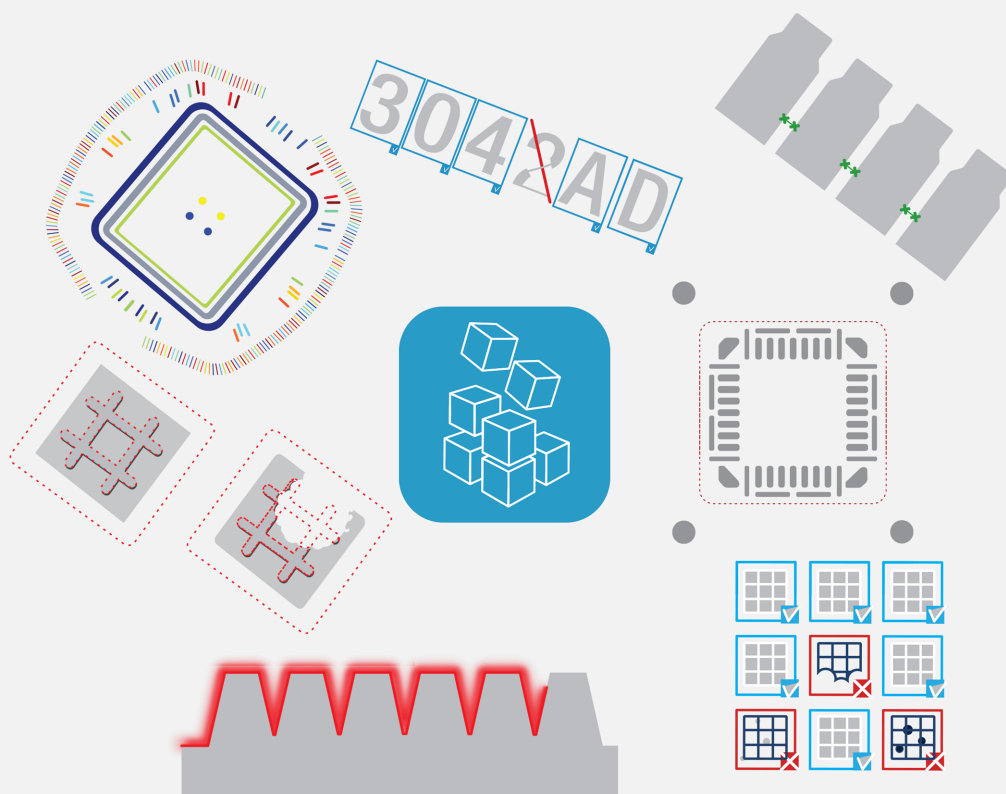




# USER GUIDE

# Open eVision

## Easy3D Compatibility with Nerian 3D Sensors



This documentation is provided with **Open eVision 24.02.0** (doc build **1196**).  
[www.euresys.com](https://www.euresys.com)

This documentation is subject to the General Terms and Conditions stated on the website of **EURESYS S.A.** and available on the webpage <https://www.euresys.com/en/Menu-Legal/Terms-conditions>. The article 10 (Limitations of Liability and Disclaimers) and article 12 (Intellectual Property Rights) are more specifically applicable.

# Easy3D Compatibility with Nerian 3D Sensors

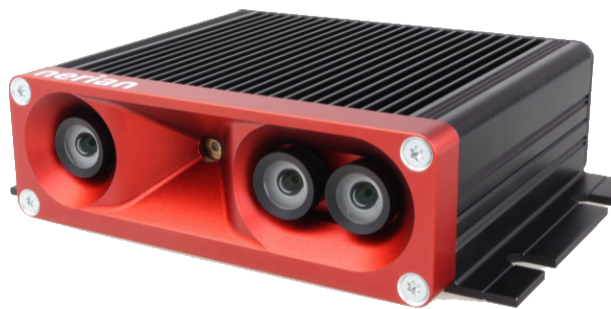
## Introduction

---

The **Nerian** 3D sensors are stereo vision-based systems.

The specifications are available on the manufacturer website:

<https://nerian.com/products>



- 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** website.

## Resources

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

- The **Nerian Ruby 3D Depth** camera (firmware 1.2.2).
- The **Nerian Vision software** 10.6.0 available at <https://nerian.com/support/software>.
- **Open eVision** 24.02
- **Microsoft Visual Studio** 2017

## Features

---

- The **Nerian Vision software** gives access to the computed disparity map in the form of a 16 bits per pixel image.
  - The lower 12 bits encode the disparity as a fixed-point value.
- It also exposes functions to convert the disparity map to 3D XYZ positions and depth maps, using the embedded calibration data.
  - Both these representations express the 3D positions in a metric space.
  - The per pixel color information is also available.
- Refer to the latest sections for code snippets showing how to transform these representations to **Easy3D** objects.

## Easy3DGrab sample application

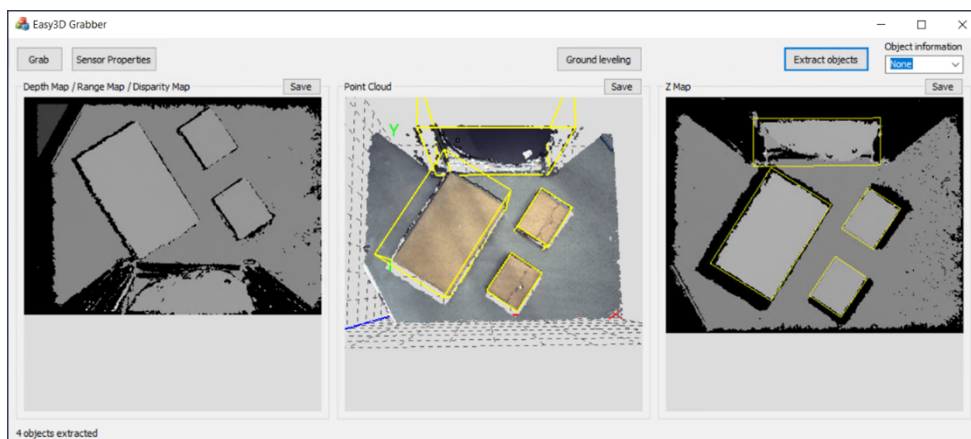
**Easy3DGrab** is distributed with C++ source code as an **Open eVision** additional resource.

It demonstrates the acquisition of **Nerian** sensors data to **Easy3D** objects.

- It features the import of:
  - The disparity map to an EDepthMap16,
  - The XYZ positions to an EPointCloud,
  - The conversion of the EPointCloud to an EZMap16.

**NOTE:** Instead of having the Z origin located at the camera, the conversion uses the maximum distance as the origin with a Z increasing towards the camera.

- You can save these representations.
- Click on the **Grab** button to acquire a new image.
- Open the **Sensor Properties** dialog to change the maximum distance. This impacts the resolution of the ZMap.
- The **Object extraction** function is exposed but you can use it only with the **Easy3DObject** license.
- You can also perform a **Ground leveling** if the scene context is compatible.



The Easy3DGrab application:  
Disparity map (left), point cloud (center), ZMap (right) with extracted objects

## C++ code sample to convert Nerian Vision software formats to Easy3D objects

### Converting a disparity map to an EDepthMap

Here is the code snippet to fill an Easy3D::EDepthMap object from an acquired ImageSet:

```
visiontransfer::ImageSet image_set;
Easy3D::EDepthMap map;

// check format
int index = image_set.getIndexOf(ImageSet::IMAGE_DISPARITY);
ImageSet::ImageFormat format = image_set.getPixelFormat(index);
if (format != ImageSet::FORMAT_12_BIT_MONO)
    throw(std::runtime_error("Unsupported image format"));

// get data
int width = image_set.getWidth();
int height = image_set.getHeight();
int stride = image_set.getRowStride(ImageSet::IMAGE_DISPARITY);
const uint8_t* pixel_data = image_set.getPixelData(index);

// setup a depth map
Easy3D::EDepthMap16& map16 = dynamic_cast<Easy3D::EDepthMap16&>(map);
map16.SetSize(width, height);
uint16_t undef = map16.GetUndefinedValue().Value;
map16.SetZResolution(1.f / 256); // the source is 8.4 fixed point format, shifted to 8.8 in the depth map

// Copy disparity data to EDepthMap8
for (int y = 0; y < height; ++y)
{
    const uint16_t* src = (const uint16_t*)(pixel_data + y * stride);
    uint16_t* dst = (uint16_t*)map16.GetBufferPtr(0, y);
    for (int x = 0; x < width; ++x)
    {
        if (src[x] >= 4095) // undefined point
            dst[x] = undef;
        else
            dst[x] = src[x] << 4; // from 12bits to 16bits values
    }
}
```

## Converting a point cloud to an EPointCloud

Here is the code snippet to fill an Easy3D::EPointCloud object from an acquired ImageSet. This snippet retrieves the colors of the images and sets them in the Easy3D::EPointCloud.

```
visiontransfer::ImageSet image_set;
visiontransfer::Reconstruct3D recon_3d;
Easy3D::EPointCloud pc;

int width = image_set.getWidth();
int height = image_set.getHeight();
int npix = width * height;
float* xyzptr = recon_3d.createPointMap(image_set);

// Push valid 3D points to an EPointCloud
// Z origin is at max distance
std::vector<Easy3D::E3DPoint> pts;
pts.reserve(npix);
pc.Clear();

ImageSet::ImageType colImg = (image_set.hasImageType(ImageSet::IMAGE_COLOR)) ?
    ImageSet::IMAGE_COLOR : ImageSet::IMAGE_LEFT;
bool colorEnabled = (image_set.hasImageType(colImg));

// Convert the color information if enabled
if (colorEnabled)
{
    std::vector<EC24A> colors;
    colors.reserve(npix);

    unsigned char* pixel = image_set.getPixelData(colImg);
    ImageSet::ImageFormat image_format = image_set.getPixelFormat(colImg);
    int colorOffset;
    switch (image_format)
    {
        case ImageSet::FORMAT_8_BIT_MONO:
            colorOffset = 1;
            break;
        case ImageSet::FORMAT_12_BIT_MONO:
            colorOffset = 2;
            break;
        case ImageSet::FORMAT_8_BIT_RGB:
            colorOffset = 3;
            break;
        default: throw std::runtime_error("Illegal pixel format");
    }

    for (int i = 0; i < npix; ++i, xyzptr += 4, pixel += colorOffset)
    {
        if (xyzptr[2] < max_distance)
        {
            pts.emplace_back(xyzptr[0], xyzptr[1], max_distance - xyzptr[2]);

            switch (image_format)
            {
                case ImageSet::FORMAT_8_BIT_MONO:
                    colors.emplace_back(*pixel, *pixel, *pixel, 255u);
                    break;
                case ImageSet::FORMAT_12_BIT_MONO:
                {
                    uint8_t colorBW8 = static_cast<uint8_t>(*reinterpret_cast<uint16_t*>(pixel) >> 4);
                    colors.emplace_back(colorBW8, colorBW8, colorBW8, 255u);
                    break;
                }
            }
        }
    }
}
```

```
        case ImageSet::FORMAT_8_BIT_RGB:
            colors.emplace_back(*pixel, pixel[1], pixel[2], 255u);
            break;
        }
    }
}

pc.AddPoints(pts);
pc.FillAttributeBuffer(static_cast<int>(Easy3D::E3DAttribute_Color), colors);
}
else
{
    for (int i = 0; i < npix; ++i, xyzptr += 4)
        if (xyzptr[2] < max_distance)
            pts.emplace_back(xyzptr[0], xyzptr[1], max_distance - xyzptr[2]);

    pc.AddPoints(pts);
}
```

### [Converting a ZMap to an EZMap](#)

Here is the code snippet to fill an Easy3D::EZMap object from an acquired Easy3D::EPointCloud:

```
Easy3D::EZMap16 zmap;
Easy3D::EPointCloudToZMapConverter converter;
converter.Convert(point_cloud, &zmap);
```