= altiro3D =

2D-to-3D image and video conversion library for free-view LCD

(starting from a single image or frame)

User's Guide

---

version 3.0 - April 2023

by E. Canessa & L. Tenze

https://github.com/canessae/altiro3D

SECTION 1

About

altiro3D is a free, extended C++ Library developed to reconstruct reality from a given single imagine (e.g., .png, .jpg) or flat video (e.g., .mp4). This is done generating a light-field (or Native) image which can be displayed in a specific light-field display such as the Looking Glass (LG) Portrait[1]. The latter is a low cost lenticular 3D device which allows to reduce loss of resolution in the horizontal direction, by slanting the structure of the lenticular lens.

altiro3D is implemented using the OpenCV Library with the goal of minimizing the image processing time to approach real time applications in 3D streaming using low-cost hardware without the need for the viewer to wear any special 3D glasses. It runs under Linux O.S.

Multiview displays (e.g., lenticular, auto stereoscopic) require multiple views of a scene to provide motion parallax and get a realistic 3D experience. By changing the viewing angle of the display, different stereoscopic pairs are perceived by the viewer. However, capturing arbitrary number of views can be cumbersome, and in some occasions impossible due to occlusion.

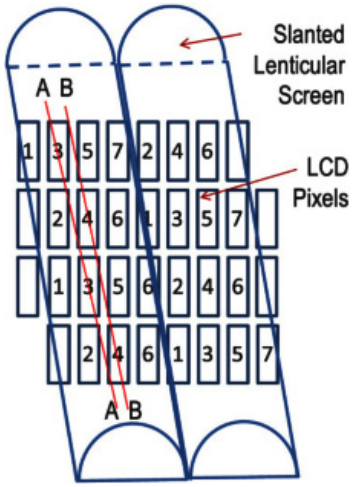

Fig.1: Slanted lenticular screen on a LCD array used to enhance image quality.[2]

This class of slanted lenticular 3D displays allows to display a hologram having N×M different views formed through a collage of images (or Quilt as in Fig.2) without the need of wearing any headsets. These devices combine light-field and volumetric technologies, and allow to visualize 3D scenarios for around 40°to 100°fields of view (FOV). These 3D screens have specific computer display calibration values each for a correct rendering.

[1] https://lookingglassfactory.com/looking-glass-portrait

[2] For details see Geng J., Adv. Opt. and Photon. Vol.5 (2013) pp. 456-535

Fig.2: Left: Each 2D Quilt tile contains sequential views of a scene, starting from the bottom-left tile as the input image. Right: section of the Quilt (6×8).

The CPU time for the computer rendering of a Quilt, and especially the holographic multiview outputs (i.e., figure 3), varies considerably between different interpolation algorithms used to obtain a virtual translation motion between consecutive virtual images. These include an interpolation for the approximate neighborhood pixel intensities, warping, optimization, and the in-painting of occlusions (i.e., black spots, out-of-plane movements) to get an acceptable illusion of depth and parallax in the horizontal direction. Here, OpenCV in-painting techniques are used[3].
A compressed Quilt allows for an efficient way to store N-view images or frames from a video. Quilts serve to save and retrieve images displayed in a LG Portrait.
The N×M images forming the Quilt, are converted into a light-field image via the following expression for the relation between the pixels of a slanted lenticular 3D LCD and the multiple perspective views[4].

$$N_{i,j} = N_{tot}(i - i_{off} - 3_j tan(\alpha))mod(P_x)/P \quad , \tag{1}$$

where i and j denote the panel coordinates for each sub-pixel. Each sub-pixel on the free-view LCD is mapped to a certain view number and color value (i.e., in the light-field domain). N denotes the view number of a certain viewpoint, α the slanted angle between the lenticular lens and the 3D LCD panel and P_x the lenticular pitch. Scaling multiple views (e.g., scaling from the Quilt to the Native image) requires lots of CPU resources and increases system complexity. With the generation of a device-dependent LUT table[5] this time can be reduced considerably.[6]
The altiro3D Library (*ref.*[7] *'altiro'*) creates multiview images via Eq.(1) starting from a monocular scene (or, alternatively, from a given set of sequential photos) to form the

[3]https://docs.opencv.org/3.4/df/d3d/tutorial_py_inpainting.html
[4]see van Berkel C., Proc. SPIE 3639, Stereoscopic Displays and Virtual Reality Sys. VI, 1999.
[5]https://en.wikipedia.org/wiki/Lookup_table
[6]see Canessa E., Tenze L., Proc. IS&T Int'l. Symp. on Electronic Imaging: Stereoscopic Displays and Applications XXXI, 53-1, 2020
[7]Real Academia Española: adv. coloq. Chile. 'inmediately' (to the point, fast).

Native light-field image. The intermediate views generated using altiro3D are then added into a Quilt collage sequentially from a given number of computed in-between snapshots. To achieve this, altiro3D (*i*) takes into account display calibration data for each specific 3D monitor, (*ii*) uses the MiDaS 2.1 deep neural network (DNN) di OpenCV algorithm[8] for a robust monocular depthmap estimation, and (*iii*) implements a one-time configuration LUT as a simple array indexing operations that save runtime computation. Although the MiDaS 2.1 '*small*' model (included in altiro3D code) cannot provide complete depth information on distant regions, the generation of multiview images starting from a single image (or video frame) through altiro3D can still offer a potential alternative method for fast 3D vision. Better results can be found using MiDaS 2.1 hybrid and large models, but these require extensive computations for the conversion and they present limits for any real-time 3D streaming. Hence altiro3D via MiDaS models, extrapolates the parallax encoded in the synthetic depthmap from monocular images (or video frames) by retrieving information from nearby pixels only.

Fig.3: Example of multiview Native output by altiro3D Library (right) starting from the given image (left).

1.1 Virtual Camera's N-Viewpoints

Within altiro3D, a given number of intermediate virtual views (having different viewing angles needed to create Native holographic frames from Quilts) are generated as follows. By default, a "FAST" algorithm has been implemented to handle 3D projecting camera and scene transformations along N-viewpoints. It uses the degree of depth to move proportionally the pixels, assuming the original image to be at the center of all the viewpoints. This is achieved with the '`cv::remap`' command of OpenCV by taking pixels from one place in the image and locating them in another position in a

[8]https://github.com/isl-org/MiDaS

new image. This "FAST" technique gives reasonable virtual interpretations of reality –at least, within a wide FOV.
For the generation of multiview images giving as input an original camera's RGB image, altiro3D algorithm can also be used with the Depth Image Based Rendering (DIBR) algorithm[9]. DIBR allows to synthesize N number of virtual images from the (almost equivalent) "REAL" camera N geometric viewpoints between the original camera and the virtual camera (i.e., using the '`-g`' option). It requires to calibrate a priori several intrinsic and extrinsic camera parameters[10]. Although this algorithm can give realistic results for single input images, it is both complex and slow, making it computational extensive for real-time 3D video streaming.

1.2 Features

In brief, the several altiro3D command lines provide the following features:

- Create Native (i.e., 3D image) from Photo using MiDaS-small.
- Create Native from Photo using MiDaS-large.
- Create Native from a given original camera's RGB image and depth image.
- Create Native from a given Quilt (N×M).
- Create Native from sorted N-views (i.e., sequential set of plain images) stored in a given directory.
- Convert Quilt views to 2D video (.mp4).
- Convert given 2D video to Native 3D video (.mp4).

All these forms of Native images/videos consisting of light-fields can be correctly displayed on a specific slanted, lenticular screen.
Most of these procedure with the "REAL" method for obtaining geometric viewpoints between the original and virtual camera (i.e., using the '`-g`' option), can require some calculation power, since the final Native image must have a resolution of 3360x3360 px with RGB color channels for a 6x8 Quilt –such that, once the pixel to be mapped is fixed, the map value for each color channel implies separated calculations. In essence this procedure with the '`-g`' option as such makes real-time video in 3D difficult to achieve.
However, since this mapping matrix depends on the geometric position of each pixel, and on the calibration parameters of the lenticular display, the constructed Lookup Tables (LUT) by altiro3D allows to reduce considerably the runtime computation to save processing time. This LUT is created by altiro3D only once at the beginning of the mapping process:

[9] https://github.com/3ZadeSSG/DIBR-Algorithm
[10] https://ftp.cs.toronto.edu/pub/psala/VM/camera-parameters.pdf

```
LCD device-related parameters
      → LUT
           → real image input
                → Quilt
                     → Native:  3D image or 3D video
                          → Display on a free-view LCD device
```

This process is repeated for any Native image/frames being visualized.

1.3 Copyright, Credits and Contacts

The free altiro3D C++ Library for the image and video conversion of free-view LCD devices is developed within the Science Dissemination Initiative of the ICTP, Trieste, Italy. For further information, binaries, papers, presentations, manuals, or to report Bugs, please visit our project website: https://github.com/canessae/altiro3D

Requirements

The simplest hardware needed to implement altiro3D is a standard PC Computer (Intel Core i5, 64bit and at least 4G RAM), running a recent release of Linux O.S. (22.04LT or newer) and any slanted lenticular display such as the LG Portrait in the figure. This is an external HDMI video monitor which provides a novel glasses-free way to preview 3D objects and scenes within a FOV 40°to 100°.

Fig.4: Headset-free Looking Glass 3D Protrait.[11]

Each LG Portrait holds its own calibration ('`visual.json`') data for a correct visual rendering.

- It is necessary to copy FIRST the file: '`visual.json`' (containing the serial Port code '`LKG-Pxxxxx`' from '`/media/<username>/LKG-Pxxxxx/LKG_calibration`' into '`bin/`'.

- Use '`altiro3D/bin/visual.json`' calibration file to create a LUT map: '`portrait-Y×X.map`' for this particular light-field display

• To improve details on the computed depthmap, it would be also necessary to download first '`model-f6b98070.onnx`'[12] (397 MB) and to add it to the directory '`models/`'.[13]

[11] `www.lookingglassfactory.com`

[12] ONNX: Open Neural Network Exchange is an open standard to represent machine learning models

[13] Use, e.g., '`wget https://github.com/isl-org/MiDaS/releases/download/v2_1/model-f6b98070.onnx`'

SECTION 3

Install

The latest version of the Debian '`altiro3D-x.x.x-Linux.deb`' package can be downloaded from https://github.com/canessae/altiro3D

3.1 Dependencies

It is necessary to install first some extra packages and their dependencies. The following packages (and their dependencies) are needed to be installed before hand:

libopencv-videoio4.5d, libopencv-imgproc4.5d, libopencv-core4.5d, libopencv-imgcodecs4.5d, libopencv-contrib4.5d, libopencv-calib3d4.5d, libopencv-imgcodecs4.5d, libopencv-video4.5d, libopencv-flann4.5d, libopencv-dnn4.5d, libqt5widgets5, libqt5gui5, libqt5core5a, libhidapi-libusb0

The updated list of required packages can be found as

```
dpkg -I altiro3D-x.x.x-Linux.deb
```

To install the packages listed above, issue the command:
'`sudo apt-get install <pkg1> <pkg2> ···`' and so on. For example,

```
sudo apt-get install libopencv-videoio4.5d libopencv-imgproc4.5d
                ...    ...
```

3.2 Library Install

To install the altiro3D '(`.deb`') package in the '`/opt`' directory type the command

```
sudo dpkg -i altiro3D-x.x.x-Linux.deb
```

In case some '`warning: files list file for package ···`' may appear on the screen after having issued '`dpkg`', then type

```
sudo apt-get install -f
```

which install what is missing to '`dpkg`'.
It also necessary to set the path to the working directory before the altiro3D commands can be used. Then, type:

```
cd /opt/altiro3D/bin/          << NOTE: the capitol "D"
source setupvars.sh
```

which defines

```
INSTALLDIR="/opt/altiro3D"
export PATH=$INSTALLDIR/bin:${PATH}
export LD_LIBRARY_PATH=$INSTALLDIR/lib:${LD_LIBRARY_PATH}
```

3.3 Uninstall

In order to remove the altiro3D ('`.deb`') package type

```
sudo dpkg -r altiro3D
```

SECTION 4

Needed Commands!

To start using the altiro3D line commands, some important comments are necessary.

By default, a ”FAST” algorithm has been implemented within altiro3D to handle 3D projecting camera and scene transformations along N-viewpoints. It uses the degree of depth to move proportionally the pixels, assuming the original image to be at the center of all the viewpoints.
For the generation of multiview images giving as input an original camera's RGB image, altiro3D algorithm can be used with the '`-g`' option. Although this option can give ”REAL”-listic results for single input images, it is both complex to configure and slower.
Few examples on the use of altiro3D commands are included in the directory '`bin/examples`'.

4.1 Retrieve Calibration Data from a Specific 3D Display

As mentioned, each LG Portrait holds a unique per-device calibration ('`visual.json`') data for a correct visual rendering. This is set in the phase of manufacturing!

It is necessary to copy FIRST the file: '`visual.json`' (containing the serial Port code '`LKG-Pxxxxx`' from '`/media/<username>/LKG-Pxxxxx/LKG_calibration`' into '`bin/`'.

Issue then the command

```
cp /media/<username>/LKG-Pxxxxx/LKG_calibration/visual.json
                bin/
```

with '`LKG-Pxxxxx`' being your own one.
For example, the '`visual.json`' usually looks like this:

{ ”configVersion”: ”1.0”, ”serial”: ”LKG-PORT-19930”, ”pitch”: { ”value”: 52.58429360652222 }, ”slope”: { ”value”: -7.179160974003201 }, ”center”: { ”value”: 0.46257855394500685 }, ”fringe”: { ”value”: 0.0 }, ”viewCone”: { ”value”: 40.0 }, ”invView”: { ”value”: 1 }, ”verticalAngle”: { ”value”: 0 }, ”DPI”: { ”value”: 324.0 }, ”screenW”: { ”value”: 1536.0 }, ”screenH”: { ”value”: 2048.0 }, ”flipImageX”: { ”value”: 0 }, ”flipImageY”: { ”value”: 0 }, ”flipSubp”: { ”value”: 0.0 } }

4.2 Create LUT Map from Calibration File

Using your own '`visual.json`' calibration file, create a LUT map: '`"portrait-Y×X.map"`' for your particular light-field Portrait based on a Quilt 6x8 (size 3360x3360 px).

```
./altiro3Dnative -m portrait-6x8.map -q 6x8 -r 560x420
                visual.json
```

In this example, the binary '`portrait-6x8.map`' file contains the LUT mapping for a default 6x8 Quilt whose single image is assumed to have a resolution 560x420 px. This '`.map`' file can be modified for different Quilt sizes (e.g., 5x9), adapted to the calibration '`visual.json`' file. The mapping procedure needs to be very accurate according to the Quilt dimensions and has to take into account the specific display calibration data.

Alternative Commands

5.1 Create Native (i.e., 3D image) from Photo using MiDaS-small

The MiDaS 2.1 '*small*' network (included in the directory '`models/`') cannot provide complete depth information on distant regions, the generation of multiview images starting from a single image (or video frame) can still give reasonable results.

- Input: Photo, MiDaS small, FAST (default)

```
./altiro3Dimage -n ../models/model-small.onnx
                -m portrait-6x8.map -q 6x8 -r 560x420
                                sample.jpg native_sample.png
```

- Input: Photo, MiDaS small, REAL '`-g`' (–geometric) option: Using real geometric equations.

```
./altiro3Dimage -g -n ../models/model-small.onnx
                -m portrait-6x8.map -q 6x8 -r 560x420
                                sample.jpg native_sample-g.png
```

- Input: Photo, MiDaS small, FAST, '`-k`' option: To resize input image to 2048*1536 px (same as the output of the LG Portrait). With this option the Native 3D may improve details on the far depth areas.

```
./altiro3Dimage -k -n ../models/model-small.onnx
                -m portrait-6x8.map -q 6x8 -r 560x420
                                sample.jpg native_sample-k.png
```

- Input: Photo, MiDaS small, REAL '`-g`' and '`-k`' options

```
./altiro3Dimage -k -g -n ../models/model-small.onnx
                -m portrait-6x8.map -q 6x8 -r 560x420
                                sample.jpg native_sample-g-k.png
```

```
[ WARN:0] global ./modules/dnn/src/dnn.cpp (1447) setUpNet DNN module was not built with CUDA backend; switched to CPU
```

5.2 Create Native from Photo using MiDaS-large

To improve details on the computed depthmap, it would be also necessary to download first '`model-f6b98070.onnx`' (397 MB) from '`https://github.com/isl-org/MiDaS`' and to add it to the directory '`models/`'.[14]

[14] Use, e.g., '`wget https://github.com/isl-org/MiDaS/releases/download/v2_1/model-f6b98070.onnx`'

Better results can be found using MiDaS 2.1 large models (via the '`-l`' option, but note that these require extensive computations.

- Input: Photo, MiDaS large, FAST (default)

```
./altiro3Dimage -l -n ../models/model-f6b98070.onnx
                -m portrait-6x8.map -q 6x8 -r 560x420
                                sample.jpg native_sample-l.png
```

Together with '`-l`', you can also add any of the other line command options here too, e.g., '`-g`', '`-k`' or both.

5.3 Create Native from a given original camera's RGB image and depth image

In this case, you need to use the '`-d`' (–depthmode) option, as follows

- Input: Photo + Depthmap, FAST

```
./altiro3Dimage -d -m portrait-6x8.map -q 6x8
                -r 560x420 sample.jpg sample-depthmap.png
                                native_sample-d.png
```

Together with '`-d`', you can also add any of the other line command options here too, e.g., '`-g`', '`-k`' or both.

5.4 Create Native from a given Quilt (N×M)

The following commands create a Native image starting from different given Quilts. Each tile in the Quilt must be ordered as shown in Fig.2.

- Input: Quilt 6x8 (size 3360x3360 px)

```
./altiro3Dimage -m portrait-6x8.map -q 6x8 -r 560x420
                sample-6x8-quilt.jpg native_sample-6x8-quilt.png
```

- Input: for Quilt 9x5 (larger size 4095x4095 px)

In this example, the binary '`portrait-9x5.map`' file contains the LUT mapping for a default 9x5 Quilt whose single image is assumed to have a resolution 455x819 px, is adapted to the calibration '`visual.json`' file.

```
./altiro3Dnative -m portrait-9x5.map -q 9x5 -r 455x819
                visual.json
```

Then, issue the command

```
./altiro3Dimage -m portrait-9x5.map -q 9x5 -r 455x819
                sample-9x5-quilt.jpg native_sample-9x5-quilt.png
```

5.5 Create Native from sorted N-views (i.e., sequential set of plain images) stored in a given directory.

If you have N consecutive views in the directory '`dir-6x8-views/`', use first '`mogrify -resize 420x560!  *.jpg`' to create an equivalent 6x8 Quilt (size 3360x3360 px).

Then, type

```
./images2native -m portrait-6x8.map -q 6x8
                dir-6x8-views/ native_from-views-6x8.png
```

For a 9x5 Quilt (size 4095x4095 px) use '`mogrify -resize 455x819!  *.jpg`', and generate the '`portrait-9x5.map`' as indicated in Section 5.4 above. Then, type

```
./images2native -m portrait-9x5.map -q 9x5
                dir-9x5-views/ native_from-views-9x5.png
```

5.6 Convert Quilt views to 2D video (.mp4)

The option '`--lg`' (–looking <video filename>) allows to create a video from quilt views. Note that this option requires two '`--`' signs!

```
./altiro3Dimage -n ../models/model-small.onnx
     -m portrait-6x8.map -q 6x8 -r 560x420
          sample.jpg native_sample.png --lg sample-2Dvideo.mp4
```

You can also add any of the other line command options here too, e.g., '`-g`', '`-k`' and/or '`-l`'. (Note that with '`-l`', it is necessary to download first the MiDaS large network '`model-f6b98070.onnx`' as indicated in Section 5.4 above).
To reduce scene deformations/distortions, you can discard the first and last seconds of the Native video

```
ffmpeg -ss 00:00:01 -i sample-2Dvideo.mp4 -to 00:00:05
                sample-2Dvideo-trim_1-5.mp4
```

5.7 Convert given 2D video to Native 3D video (.mp4)

A pre-recorded flat video (e.g., .mp4 file) can be converted into a Native 3D Video.

```
./altiro3Dvideo -n ../models/model-small.onnx
          -m portrait-6x8.map -q 6x8 -r 560x420
                   sample-2Dvideo.mp4 sample-3Dvideo_noAudio.mp4
```

To convert 2D video to 3D video takes some time! specially when adding the options: '`-g`', '`-k`' and/or '`-l`' (recall that with '`-l`', it is necessary to download first the MiDaS large network '`model-f6b98070.onnx`' as indicated in Section 5.4 above). However, it can be useful to carry out this exercise for short videos or larger, just for testing purposes.

5.7.1 Add audio tracks to 3D Native video

altiro3D uses a '`ffmpeg`'-based script to add the audio tracks to the converted 3D videos. This is found in the '`scripts/`' directory and can be used like this:

```
./ffmpeg_audio_extract_and_merge.sh
          <input-2Dvideo_withAudio> <video3D_noAudio>
```

Note that the '`video3D_noAudio`' will preserve the name BUT will contain the audio tracks. For example, to complete the conversion from '`sample-2Dvideo.mp4`' to Native '`sample-3Dvideo.mp4`' use

```
./ffmpeg_audio_extract_and_merge.sh
          sample-2Dvideo.mp4 sample-3Dvideo_noAudio.mp4

and then change filename

mv sample-3Dvideo_noAudio.mp4 sample-3Dvideo.mp4
```