**ISO/IEC JTC 1/SC 29/WG 4**

**MPEG Video Coding**

**Convenorship: CN**

| | |
|---|---|
| **Document type:** | Output Document |
| **Title:** | Manual of Immersive Video Depth Estimation 3 |
| **Status:** | Approved |
| **Date of document:** | 2021-01-29 |
| **Source:** | ISO/IEC JTC 1/SC 29/WG 4 |
| **Expected action:** | None |
| **Action due date:** | None |
| **No. of pages:** | 11 |
| **Email of Convenor:** | yul@zju.edu.cn |
| **Committee URL:** | https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg4 |

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**
**ORGANISATION INTERNATIONALE DE NORMALISATION**
**ISO/IEC JTC 1/SC 29/WG 4 MPEG VIDEO CODING**

**ISO/IEC JTC 1/SC 29/WG 4 N 0058**
**January 2021, Online**

| | |
|---|---|
| **Title** | **Manual of Immersive Video Depth Estimation 3** |
| **Source** | **WG 4, MPEG Video Coding** |
| **Status** | **Approved** |
| **Serial Number** | **20010** |

# 1 Introduction

This document describes the third version of depth estimation technique and software called Immersive Video Depth Estimation (IVDE), which addresses depth estimation from video acquired by multiple omnidirectional or perspective cameras, needed to create multi-point 6DoF/3DoF+ scene representation.

# 2 Table of contents

1 Introduction ..... 1
2 Table of contents ..... 1
3 IVDE ..... 2
3.1 Depth estimation ..... 2
3.2 Neighboring segments depth analysis ..... 5
3.3 Temporal consistency enhancement ..... 5
3.4 Parallelization of graph-based optimization ..... 6
3.5 Segmentation in omnidirectional videos ..... 7
3.6 Automatic depth range calculation ..... 8
3.7 Handling of the encoder-derived features ..... 9
4 Building the project ..... 9
4.1.1 Using the command line (e.g. Unix) ..... 9
4.1.2 Using a GUI (e.g. Windows) ..... 9
5 Instructions to run IVDE ..... 10
5.1.1 Estimation parameters ..... 10
5.1.2 Sequence parameters ..... 10
5.1.3 Filenames ..... 11
6 MPEG Repository ..... 11
7 References ..... 11

# 3 IVDE

The framework is based mainly on the method described in [1]. The particular usefulness of the presented method in virtual navigation, free-viewpoint television and other 6DoF systems, is a result of the joint exploitation of the ideas mentioned below:

- Depth is estimated for segments instead of individual pixels, and thus the **size of segments can be used to control the trade-off between the quality of depth maps and the processing time of estimation**. Larger segments can be used to attain fast depth estimation, or finer segments can be used to attain higher quality.
- **Estimation is performed for all views simultaneously and produces depths that are inter-view consistent** because of the utilization of the new formulation of the cost function, developed for segment-based estimation.
- No assumptions about the positioning of views are stated **and any number of arbitrarily positioned cameras (both perspective and omnidirectional)** can be used during the estimation.
- In the temporal consistency enhancement method, **depth maps estimated in previous frames are utilized in the estimation of depth for the current frame,** increasing the consistency of depth maps and simultaneously decreasing the processing time of estimation.
- The framework uses a parallelization method that **reduces the processing time of graph-based depth estimation.**

## 3.1 *Depth estimation*

The estimation of depth in the proposed method is based on a cost function minimization, performed using GraphCut method [2]. The cost function is based on two components: the intra-view discontinuity cost $V_{s,t}$ and the inter-view matching cost $M_{s,s'}$, responsible for the inter-view consistency of depth maps:

$$E(\underline{\mathrm{d}}) = \sum_{c \in \mathrm{C}} \sum_{s \in \mathrm{S}} \left\{ \sum_{c' \in \mathrm{D}} M_{s,s'}(d_s) + \sum_{t \in \mathrm{T}} V_{s,t}(d_s, d_t) \right\},$$

where:

$\underline{\mathrm{d}}$ – vector containing depth value for each segment in all views,
C – set of views,
$c$ – view used in the estimation,
D – set of views neighboring to the view $c$,
$c'$ – view neighboring to the view $c$,
S – set of segments of the view $c$,
$s$ – segment in the view $c$,
$d_s$ – currently considered depth of the segment $s$, $d_s \in \underline{\mathrm{d}}$,
$s'$ – segment in the view $c'$, which corresponds to the segment $s$ in the view $c$ for the currently considered depth $d_s$,

$M_{s,s'}$ – inter-view matching cost between segments $s$ and $s'$,
T – set of segments neighboring to the segment $s$,
$t$ – segment neighboring to the segment $s$,
$V_{s,t}$ – intra-view discontinuity cost between segments $s$ and $t$,
$d_t$ – currently considered depth of the segment $t$, $d_s \in \underline{\mathrm{d}}$ .

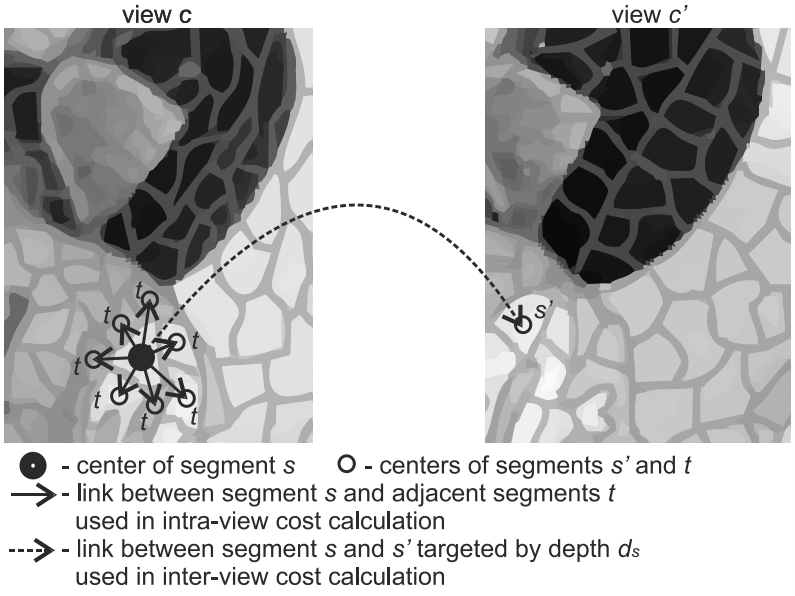



*Fig. 1. Inter-view and intra-view costs.*

The intra-view discontinuity cost is calculated between all neighboring segments within the same view:

$$V_{s,t}(d_s, d_t) = \beta \cdot |d_s - d_t| ,$$

where:

$\beta$ – smoothing coefficient,
$d_s$ – currently considered depth of the segment $s$,
$s$ – segment in the view $c$,
$t$ – segment neighboring to the segment $s$,
$V_{s,t}$ – intra-view discontinuity cost between segments $s$ and $t$,
$d_t$ – currently considered depth of the segment $t$.

In the proposed method the smoothing coefficient $\beta$ is not fixed for all segments. Instead, the smoothing coefficient is calculated using a similarity of two neighbouring segments *s* and *t* and $\beta_0$ that is an initial smoothing coefficient:

$$\beta = \beta_0 / \left\| [\hat{Y}\ \hat{C}_b\ \hat{C}_r]_s - [\hat{Y}\ \hat{C}_b\ \hat{C}_r]_t \right\|_1 ,$$

where:

$\beta$ – smoothing coefficient,

$\beta_0$ – initial smoothing coefficient provided by the user,

$\|\cdot\|_1$ – L1 distance,

$s$ – segment in the view $c$,

$t$ – segment neighbouring to the segment $s$,

$[\hat{Y}\ \hat{C}_b\ \hat{C}_r]_s$ – vector of average $Y$, $C_b$, $C_r$ color components of the segment $s$,

$[\hat{Y}\ \hat{C}_b\ \hat{C}_r]_t$ – vector of average $Y$, $C_b$, $C_r$ color components of the segment $t$.

The core of the inter-view matching cost, denoted as $m_{s,s\prime}$, is:

$$m_{s,s\prime}(d_s) = \frac{1}{count(\mathrm{W})} \sum_{w \in \mathrm{W}} \left\| [Y C_b C_r]_{\mu_s + w} - [Y C_b C_r]_{T[\mu_s]+w} \right\|_1 ,$$

where:

W – set of points in the window of the size specified by the user,

count(·) – size of the window W,

$w$ – vector of coordinates of a point in the window W,

$\|\cdot\|_1$ – L1 distance,

$\mu_s$ – vector of coordinates of center of a segment $s$,

$T[\cdot]$ – 3D transform obtained from intrinsic and extrinsic parameters of cameras,

$[Y\ C_b\ C_r]_{\mu_s+w}$ – vector of $Y$, $C_b$, $C_r$ color components of the center $\mu_s$ of the segment $s$,

$[Y\ C_b\ C_r]_{T[\mu_s]+w}$ – vector of $Y$, $C_b$, $C_r$ color components of the point in a view $c'$ corresponding to the center $\mu_s$ of the segment $s$ in a view $c$.

In order to achieve the inter-view consistency of depth maps, the value of the inter-view matching cost $M_{s,s'}(d_s)$ is calculated as [2]:

$$M_{s,s'}(d_s) = \begin{cases} \min\{0, m_{s,s'}(d_s) - K\} & if\ d_s = d_{s\prime} \\ 0 & if\ d_s \neq d_{s\prime} \end{cases} ,$$

where:

$s$ – segment in the view $c$,

$d_s$ – currently considered depth of the segment $s$,

$s'$ – segment in the view $c'$, which corresponds to the segment $s$ in the view $c$ for the currently considered depth $d_s$ ,

$d_{s\prime}$ – currently considered depth of the segment $s'$,

$M_{s,s'}$ – inter-view matching cost between segments $s$ and $s'$,

$m_{s,s'}$ – core of the inter-view matching cost between segments $s$ and $s'$,

$K$ – a positive constant.

The value of constant $K$ is selected so that the inter-view matching cost $M_{s,s'}$ is not dominated by the intra-view discontinuity cost $V_{s,t}$ , as a sum of these two costs constitutes the cost function of

the depth optimisation. The chosen final value of $K$ is 30, as discussed in [1]. The use of both equirectangular and perspective views is included in the 3D transform $T[\cdot]$.

## 3.2 *Neighboring segments depth analysis*

In order to increase the final quality of estimated depth maps, a segment-based method of the depth enhancement, named neighboring segments depth analysis, was included.

The proposed process is performed for each segment in estimated depth maps. For the currently processed segment, depth values of its neighboring segments are tested as new depth candidates for this segment. A depth value is used if two conditions are fulfilled: use of this depth reduces the inter-view matching cost for the processed segment and a corresponding segment in neighboring view targeted by this depth also has the same value of depth.

The proposed solution increases the quality of depth maps in areas of uncertain depth (e.g., disoccluded areas) and preserves the inter-view consistency of depth maps. Moreover, because the process is performed after estimating the depth for each frame, such enhanced depth is used for all following frames (because of segmentation-based temporal enhancement). Therefore, such an approach increases the quality of depth maps also in terms of temporal consistency.

## 3.3 *Temporal consistency enhancement*

In natural video sequences, only a small part of an acquired scene considerably changes in consecutive frames, especially when cameras are not moving during the acquisition of video. The idea of the proposed temporal consistency enhancement of depth estimation is to calculate a new value of depth only for the segments that changed (in terms of their color) in comparison with the previous frame.

The proposed temporal consistency enhancement method allows us to automatically mark segments as unchanged in consecutive frames. These segments are used in the calculation of the intra-view discontinuity and the inter-view matching cost for other segments, but are not represented by any node in the structure of the optimized graph. It reduces the number of nodes in the graph, making the optimization process significantly faster, and on the other hand, increases the temporal consistency of estimated depth maps.

In the first frame of a depth map, denoted as an "I-type" depth frame, the estimation is performed for all segments, as described in the previous sections. The following frames ("P-type" depth frames) can utilize depth information from the preceding P-type depth frame and the I-type depth frame.

Segment $s$ is marked by the algorithm as unchanged in two cases: if all components of the vector $[\hat{Y}\ \hat{C}_b\ \hat{C}_r]_s$ of average $Y$, $C_b$ and $C_r$ color components changed less than the set threshold $T$ in comparison with segment $s_B$, which is a collocated segment in the previous P-type frame, or, if all components of the abovementioned vector changed less than the threshold $T$ in comparison with segment $s_I$ – a collocated segment in the I-type frame. If any of these two conditions are met, then segment $s$ adopts the depth from the segment $s_B$ or $s_I$ (depending on which condition was fulfilled).

A collocated segment in the previous or the first frame is simply the segment which contains the central point of the segment $s$. Therefore, even if the segmentation in compared frames is not the same, the algorithm can easily find the corresponding segment in these frames.

The introduction of two reference depth frames has a beneficial impact on the visual quality of virtual navigation. First, the adoption of depth from the previous P-type depth frame allows us to

use the depth of objects that changed their position over time. On the other hand, the adoption of depth from the I-type depth frame minimizes the flickering of depth in the background.

## 3.4 *Parallelization of graph-based optimization*

In our proposal, each of $n$ threads estimates a depth map with an $n$-times lower number of depth levels. Depth maps with a reduced number of depth levels that were calculated by different threads have to be merged into one depth map. The merging process is performed in a similar way as depth estimation [using the cost function (1)], but only two levels of depth are considered for each segment – i.e., the depth of a segment from thread $t$ or the depth from thread $t + 1$ (Fig. 3). Only two depth maps can be merged into one by one thread during the merging cycle. Therefore, for $n$ threads, $\lceil log2(n) \rceil$ additional cycles are needed to estimate the final depth map with all depth levels.

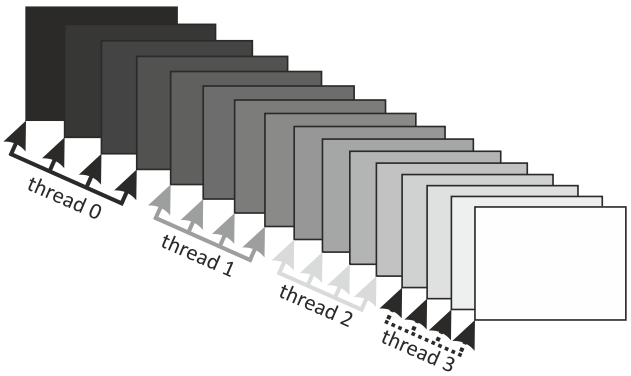



*Fig. 2. Depth levels are divided into blocks, each rectangle represents a different level of the depth of a scene.*

Of course, even without the use of parallelization, all cores of the CPU can also be used for depth estimation, e.g., each core can perform the estimation of depth for different sets of input views (e.g., for each 5 cameras of the system), or for different frames of the sequence. Unfortunately, when many standalone depth estimation processes are performed, it results in the loss of inter-view consistency or temporal consistency of estimated depth maps. When the proposed parallelization is used, both inter-view and temporal consistency of depth maps, which are fundamental for the quality of virtual view synthesis, are preserved.

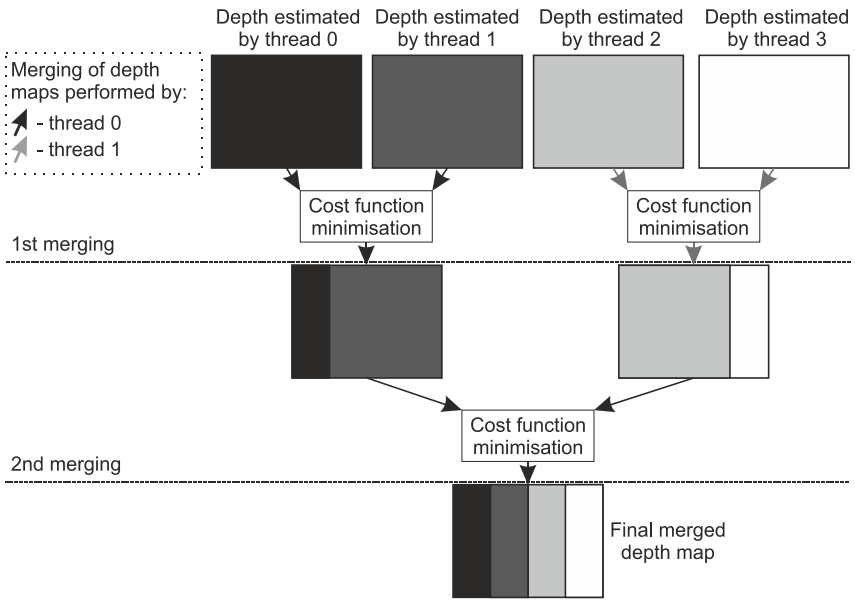



*Fig. 3. Depth map merging process for the case of 4-thread parallelization.*

## 3.5 *Segmentation in omnidirectional videos*

The use of omnidirectional cameras is taken into account during the superpixel segmentation of input views. The superpixel segmentation [3] is based on the calculation of the color and spatial distances of a point to neighboring superpixels.

Fig. 4. shows initial grid of 1000 superpixels used in the beginning of segmentation process. To estimate such initial grid, the overall size of image is divided by the number of superpixels in order to acquire the average size of superpixel. Then, the square root of the resulting superpixel size is used to define the distance between centers of superpixels and, in the end, the whole image is divided evenly as presented in the figure below.

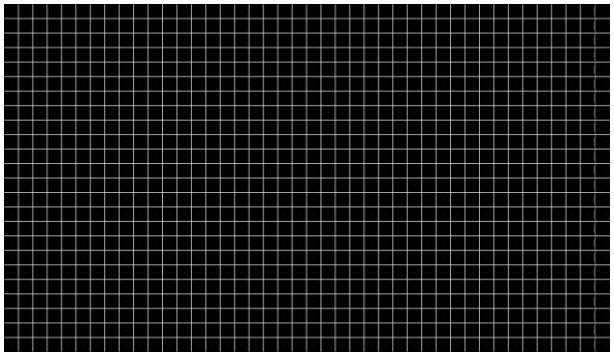


*Fig. 4. Initial grid of superpixels used in segmentation.*

In next steps, segments' shapes are changed on the basis of color and spatial distances of neighboring points in order to match edges present in a scene. The final segmentation of a omnidirectional sequence can be seen in Fig. 5.

Segments on the top and bottom border of presented image have similar size in the whole image. However, in equirectangular image, areas in the top and the bottom of an image represent much smaller areas of a scene than areas in the middle of an image. Therefore, if the segmentation of the image would be not adapted to the equirectangular images, then the accuracy of estimated depth maps would be not consistent in for the whole image in the proposed method.

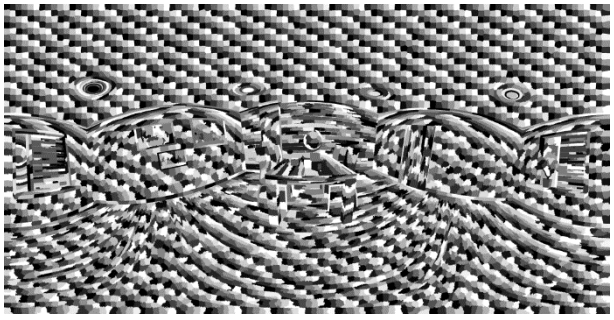


*Fig. 5. Result of unmodified superpixel segmentation for an equirectangular image.*

The initial segmentation of 360 video should be based on the equirectangular projection. First of all, as in the process of unmodified segmentation, the average distance between centers of segments is calculated as square root of the average size of a segment. This average distance is used to calculate the number of superpixels on the 'equator' (central row) of an equirectangular image. The number of superpixels in rows that are above or under the equator is proportionally lower, because these rows represent circles on a sphere that are smaller than the circle represented by the equator. The result of such initial grid of superpixels in an equirectangular image is presented in Fig. 6.

The calculation of the spatial distance in case of an omnidirectional image has to be based not simply on the difference of positions of two points in an image, but on the distance between these points before the equirectangular projection, using appropriate formulas.

The final result of such modified superpixel segmentation, adapted to equirectangular images, can be seen in Fig. 7. The size of segments in the center of an image is smaller than in unmodified superpixel segmentation, while the size of segments in the top and the bottom of an image is much larger, therefore, the proposed segmentation better represents real relative sizes of objects present in a scene.

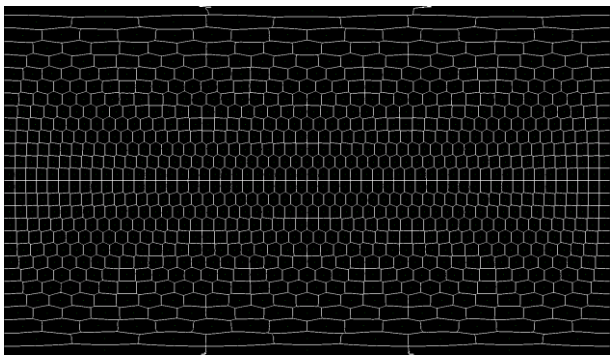


*Fig. 6. Proposed initial grid of superpixels used in segmentation of an equirectangular image.*

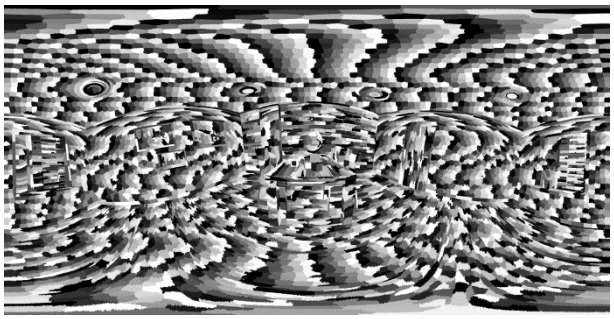
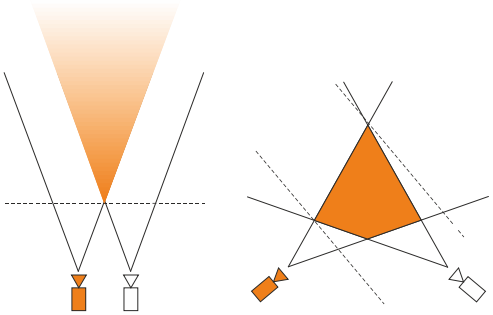


*Fig. 7. Result of modified superpixel segmentation for an equirectangular image.*

## 3.6 *Automatic depth range calculation*

In ERP content, $z_{near}$ is equal to the largest distance between views used in inter-view matching. $z_{far}$ was calculated to give minimum of 1 degree shift per pixel when the most distant views are matched.

In perspective content, the calculation of $z_{far}$ and $z_{near}$ are different for linear camera arrangement and for other arrangements.



For the orange camera, in the linear arrangement (left), the $z_{near}$ should be no closer than the $z$ for which a point on the right side of the image becomes visible on the left side in the white camera. $z_{far}$ on the other hand, can be estimated as z for which the disparity for the point in the middle of the image is no smaller than 1 pixel.

For other arrangements, the $z_{near}$ should be no closer than the $z$ for which a point on the left side of the image becomes visible on the left side in the white camera, while $z_{far}$ should be no further than the $z$ for which a point on the left side of the image becomes visible on the left side in the white camera.

To calculate the range for all possible arrangement, all abovementioned conditions are checked and, in the end, the closest calculated $z$ becomes $z_{near}$, while the most distant $z$ becomes $z_{far}$.

To reduce the calculated range to typical real-world cases, the conditions are checked with set threshold (minimum 10 pixel disparity, width/10 for matching of pixels on the edges of images). In the end, calculated $z_{near}$ and $z_{far}$ values are cropped to the range [0.3, 1000] (30 cm to 1 km).

When used, calculated depth range will be included in new sequence parameters file (see Section 5.1.1).

## 3.7 *Handling of the encoder-derived features*

In order to perform the exploration experiment on the use of the encoder-derived features in the depth estimation, the required functionalities were added to the IVDE 3.0. The new version handles $z_{near}$ and $z_{far}$ values and skip flag send from the encoder in order to increase the quality of decoder-side estimated depth maps and decrease the time of the estimation.

The example of features and configuration is included in the repository (see Section 5). For more information on the encoder-derived features see [5].

# 4 Building the project

To obtain the master branch of the project :

```
git clone http://mpegx.int-evry.fr/software/MPEG/Explorations/6DoF/IVDE.git
cd IVDE
git checkout master
```

Below are two alternative instructions for building: the first using command line tools, typically on Unix. The second instruction set is for GUI tools, typically on Windows.

### 4.1.1 Using the command line (e.g. Unix)

Configuring, building, and installing IVDE:

```
cmake -DCMAKE_BUILD_TYPE=Release
cmake --build . --config Release
make
```

### 4.1.2 Using a GUI (e.g. Windows)

Open the CMake GUI and specify:

- Where the source directory is: /Workspace/IVDE
- Where to build the binaries: /Workspace/IVDE/build
- Click Configure, Yes, Finish
- Click Generate

Build the generated project.

# 5 Instructions to run IVDE

Template configuration files are available under CTC_cfg/. The example of configuration with use of encoder-derived features is in FEATURES_cfg/. The filenames in template configuration files are examples.

To run the software, 3 JSON configuration files have to be used:

```
IVDE estimation_params.json SA_sequence_params.json SA_filenames.json
```

In case of running **MIV decoder-side depth estimating anchor**, the 'SA_sequence_params.json' is the file generated by the TMIV Decoder.

### 5.1.1 Estimation parameters

- **TotalNumberOfFrames:** Number of frames,
- **NumOfThreads:** Number of CPU threads used by software,
- **NeighboringSegmentsDepthAnalysis:** Turning on/off the neighboring segments depth analysis,
- **NumberOfZSteps:** The number of depth steps between the nearest and farthest depth planes,
- **NumberOfSuperpixels:** Number of superpixels used for the estimation of the depth map of in each view,
- **MatchNeighbors:** Number of neighboring views matched with each view,
- **MatchThresh:** The threshold of the inter-view matching cost,
- **Matcher:** Type of used matcher,
- **MatchingBlockSize:** Size of block used in inter-view matching cost,
- **NumberOfCycles:** Number of GraphCut cycles,
- **SuperpixelSegmentationType:** Type of used superpixel segmentation,
- **SuperpixelColorCoeff:** Coefficient used in superpixel segmentation to influence shapes of superpixels,
- **TemporalEnhancement:** Turning on/off the temporal consistency enhancement,
- **TemporalEnhancementIFramePeriod:** Number of frames between I-type depth frames +1,
- **TemporalEnhancementThresh:** The threshold used in the temporal consistency enhancement,
- **NumberOfCyclesInIFrame:** Number of GraphCut cycles in I-type depth frame,
- **StartFrame:** Number of first frame to be used for estimation

Optional parameters:

- **Point2BlockMatching:** Turning on/off the point to block matching method preferable when compressed input views are used,
- **AutomaticDepthRange:** Turning on/off the automatic calculation of depth range. Warning: when used, calculated depth range will be included in new sequence parameters in the file `SA_sequence_params_autoDepthRange.json`,
- **UseFeatures:** Turning on/off use of all encoder-derived features (skip flag, partition flag, zmin and zmax),
- **UseSkipFlag:** Turning on/off use of encoder-derived skip flag only,
- **BitDepthFeatures:** Number of bits per sample used in views used to extract features

### 5.1.2 Sequence parameters

Structure *cameras* should have such parameters for each view used in estimation:

- **BitDepthColor:** Number of bits per sample in an input view,

- **BitDepthDepth:** Number of bits per sample in an output depth map,
- **ColorSpace:** Color space of an input view,
- **DepthColorSpace:** Color space of an output depth map,
- **Depth_range:** The nearest and farthest depth plane in the scene,
- **Hor_range:** Horizontal range of equirectangular view (required only for equirectangular view),
- **Name:** Name of a view,
- **Position:** Position of a view,
- **Projection:** Type of a view, perspective and equirectangular views are supported,
- **Resolution:** Resolution of an input view,
- **Rotation:** Rotation of a view,
- **Ver_range:** Vertical range of equirectangular view (required only for equirectangular view)

### 5.1.3 Filenames

Structure *filenames* should have two parameters for each view used in estimation:

- **InputView:** The filename of a view,
- **OutputDepthMap:** The filename of an output depth map

Optional parameters:

- **Features:** The filename of encoder-derived features for a view

# 6 MPEG Repository

The repository for IVDE is available on MPEG GIT:

http://mpegx.int-evry.fr/software/MPEG/Explorations/6DoF/IVDE

and in the public repository:

https://gitlab.com/mpeg-i-visual/ivde

# 7 References


[1] D. Mieloch, O. Stankiewicz and M. Domański, "Depth Map Estimation for Free-Viewpoint Television and Virtual Navigation," IEEE Access, vol. 8, pp. 5760-5776, 2020.

[2] R. Achanta and S. Süsstrunk, “Superpixels and Polygons using simple non-iterative clustering,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, pp. 4895–4904.

[3] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 2, pp. 147–159, Feb. 2004.

[4] D. Mieloch, A. Dziembowski, J. Stankowski, O. Stankiewicz, M. Domański, G. Lee, J. Yun, “[MPEG-I Visual] Immersive video depth estimation”, ISO/IEC SC29/WG11 MPEG2020/M53407, Online, April 2020.

[5] P. Garus, G. Clare, F. Henry, J. Jung, “Feature-driven Decoder Side Depth Estimation”, ISO/IEC SC29/WG5, MPEG2020/M54994, Online, October 2020.