

# CAMORPH : A TOOLBOX FOR CONVERSION BETWEEN CAMERA PARAMETER CONVENTIONS

B. Brand<sup>1</sup>, M. Bätz<sup>1</sup>, J.Keinert<sup>1</sup>

<sup>1</sup> Fraunhofer Institute for Integrated Circuits IIS, Am Wolfsmantel 33, 91058 Erlangen, Germany

(benjamin.brand, michel.baetz, joachim.keinert)@iis.fraunhofer.de

## Commission II

**KEY WORDS:** Camera Parameter Conversion, Coordinate Systems, Photogrammetry, Computer Vision, Virtual Camera.

### ABSTRACT:

Defining or estimating camera poses and attributes is a vital part of computer graphics and computer vision. These areas encompass a vast amount of sub-fields like virtual reality, photogrammetry, computer generated imagery and many more. It is desirable to share this camera data between applications of these different areas to leverage their strengths in a workflow. However, a great amount of formats and conventions make it difficult to transport camera parameters from one software package to another. This paper presents an easy to use python library *camorph* to convert different formats into each other which enables the usage of a multi-software workflow for the best possible results and facilitates the comparison of different methods.

## 1. INTRODUCTION

Representations of virtual cameras are exceedingly important for many tasks in the fields of computer graphics and vision. Intrinsic camera parameters describe properties like focal length and lens distortion inherent to the camera itself (Yan et al., 2016), while extrinsic parameters describe the orientation and translation of the camera (Zhang, 2000). Many different specialized software applications like COLMAP (Schönberger, 2020), Unity (Unity Technologies, 2022), Reality Capture (Epic Games, Inc., 2010) and Blender (Blender Foundation, 2022) exist for specialized tasks. All of these different applications have varying needs for data structures and conventions. This leads to distinct formats which can be challenging to handle. In an environment where multiple software packages and file formats from different fields of application are used for research and/or development to leverage their respective strengths in a workflow, this can lead to confusion and frustration. For example, photogrammetry tools like COLMAP and Reality Capture can be used to estimate camera parameters and generate a virtual representation with a textured 3D model from pictures. Subsequently, these parameters could be used in a different specialized software to refine results for certain tasks like mesh generation, mesh refinement, texturing, rendering and more. Modern neural radiance fields (NeRF) (Mildenhall et al., 2020) like TensoRF (Chen et al., 2022) and Instant-NGP (Müller et al., 2022) typically start from already estimated or known camera poses. While there are utility scripts that convert popular calibration formats to a readable json format for these NeRF-like approaches, this is usually limited to one source format. This prevents the user from using different calibration tools, or cross-checking the quality of the image alignment in other applications where this is more convenient. To render a synthetic camera pose trace as typically done in MPEG standardization (Boyce et al., 2021) with NeRF-based approaches, the movement could be generated in Blender or any other computer graphics application, and then exported to a JSON file for rendering.

Different data structures require knowledge of the structure itself to read or convert these formats into each other. While some formats are ASCII-based and straightforward to read, others have intricate binary layouts with tree and node structures that need to be unraveled and read semantically correctly. COLMAP or NeRF-like approaches employ widely used text or json formats, whereas Unity uses a custom subset of yaml. FBX (Autodesk, Inc., 2022) implements a completely custom binary tree structure. Additionally, FBX has very little information publicly available (Blender Foundation, 2018), but is very powerful and can encompass whole 3D scenes from animations to models and cameras. Getting the correct parameters out of these data structures is often times not trivial.

Most of these applications and formats also use different coordinate systems, with different default camera orientations. Rotation can be expressed as matrices, Euler or Tait-Bryan angles, quaternions, or in yet other ways (Vince, 2011). This can lead to confusion, as equivalent representations of rotations are often times not obvious or not easily human readable, which makes it difficult to verify the correctness of the respective coordinate system conversion. Therefore, converting between different coordinate systems and camera orientations is a vital part of modern photogrammetry research to leverage the best of every software application. When those parameters are wrong, poor results in the workflow at a later point are unavoidable. Finding the source of the issue can take a long time and is also difficult to spot, which is why it is very important that those parameters are conveniently converted without the need for manual intervention.

Additionally, some intrinsic parameters can be represented in different ways, but still depict the same entity. For example, the focal length can be represented in millimeters with respect to the real-world sensor size, or in pixels with respect to the image dimensions (AirGon Support, 2018). The principal point can be represented as absolute pixel coordinates, or as an offset with respect to the sensor size in inches (Reality Capture Support,

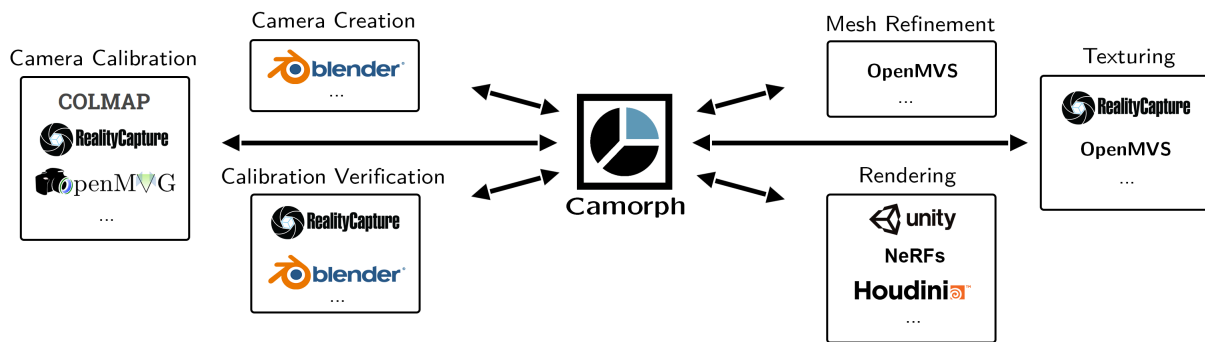


Figure 1. Potential use cases and applications for the conversion of camera parameters between formats and coordinate systems for different tasks.

2022). These different representations also behave differently when modifying the image, for example cropping or scaling. Manually converting and modifying these parameters adds unnecessary overhead to the production pipeline.

This paper presents an easy to use python library *camorph* to convert different camera parameter conventions into each other. These can then be used in different subsequent workflows: For example, cameras can be calibrated in COLMAP, and then NeRFs can be trained with this calibration. Another workflow could be creating a model in Reality Capture, refining and texturing in OpenMVS (Cernea, 2020), and then comparing to the NeRF rendering at the same camera poses. Additionally, camera pose traces can be easily created in Blender, and then exported to be rendered in another application. Table 2 is a list of all currently supported camera parameter representations in *camorph*. Section 3 will give some background information about camera parameters, while section 4 will cover implementation details of *camorph* and results will be presented in section 5.

## 2. RELATED WORK

Several families of applications can be distinguished, for example: 3D computer graphics, photogrammetry, game engines and virtual reality. While 3D computer graphics applications produce images from virtual objects, photogrammetry tools generate virtual scenes with 3D models from photographs. Game engines are specialized in creating interactive real-time renderings. Similarly, virtual reality applications need to be designed for fast rendering on VR headsets. In some cases, converting from one application to the other is straightforward. For example, when converting between 3D computer graphics applications like Blender (Blender Foundation, 2022) and Houdini (Side Effects Software Inc., 2022), both applications can read several intermediate file formats and convert accordingly. One of the most widely used format categories in this family of applications is called scene description formats (Immersive Digital Experiences Alliance, 2020c). These files usually consist of cameras, geometric objects, shaders, materials, lights, and more. Some of the most popular formats in this category are the Autodesk proprietary format FBX(Filmbox) (Autodesk, Inc., 2016) and the open source format GLTF (graphics language transmission format) (The Khronos® 3D Formats Working Group, 2021). In other application families, converting from one to another might prove more difficult. In photogrammetry for example, converting between different applications like Meshroom (Griwodz et al., 2021) to COLMAP (Schönberger, 2020) and vice versa is not straightforward, because of

how different the camera parameters are represented and stored. Additionally, converting from one software family to another, for example from Unity (Unity Technologies, 2022) to MPEG OMAF (Doré et al., 2018), is quite challenging because of the different needs of different application areas. Some plugins for applications exist that try to alleviate the problem, for example the photogrammetry importer for Blender (Blender Foundation, 2022). As this is a plug-in specifically for Blender and can only import scenes, this solution is not ideal. The source code for Local Light Field Fusion (Mildenhall et al., 2019) also provides a script to convert from the COLMAP format to the custom `poses_bounds.npy`. This format is also used in some NeRF (Mildenhall et al., 2020) variations, in addition to a custom JSON format which was generated in a Blender file provided as supplementary material. Moreover, some attempts were made to unify formats and write conversion tools like Kapture from Naver Labs (NAVER LABS Corp., 2021). While this can handle formats in the photogrammetry family, overarching support is missing.

## 3. BACKGROUND

The most widely used coordinate system is the Cartesian. It is defined by three orthogonal basis axes usually labeled  $x$ ,  $y$ , and  $z$ . However, the absolute orientation of the system, what an external viewer would call “up”, “front”, and “right” is not defined. Every application can define the orientation and handedness of these axes arbitrarily. This poses a problem with the consensus between different software packages. What a user may call “up” in one software package is not “up” in another. Therefore, if no or wrong conversion is done, the user might design a scene with the right orientation in mind, while a different software package shows the scene upside down. This is also a problem in virtual reality for example, as the orientation of the scene has to match with real directions. If there are several components from different software packages with different conventions, this has to be corrected manually, which is irritating at best and unfeasible at worst. Furthermore, some functionality of software packages depends on the right orientation of scenes and objects. For example, the Blender navigation mode is set to turntable as a default (Blender Foundation, 2022). This means the scene can not be rotated arbitrarily, but works like a record player, where the user can only rotate the viewing camera about two axes (Blender Foundation, 2022). This way, the user gets a feeling for what “up” and “down” is. This method of navigation is employed by most applications in the field of 3D computer graphics. The handedness of the system can change as well. When not accounting for this factor, scenes

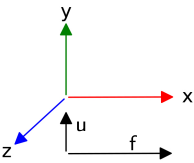
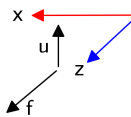
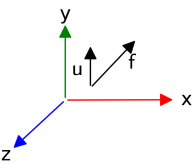
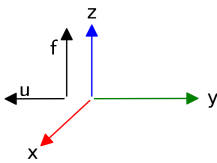
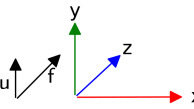
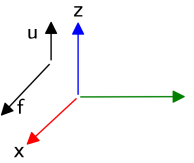
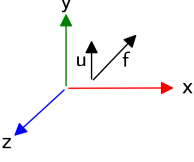
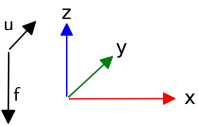
Tool/Format Name	Coordinate System	Description
FBX (Autodesk, Inc., 2022)		One of the most commonly used node-based interchange formats between 3D computer graphics software developed by <i>Autodesk</i> . Uses a custom binary node-based format.
COLMAP (Schönberger et al., 2016) (Schönberger and Frahm, 2016)		A 3D reconstruction software developed by Johannes L. Schönberger with a custom ASCII based file format for camera parameters. Uses two files for intrinsic and extrinsic camera parameters.
Meshroom (Griwodz et al., 2021)		A 3D reconstruction software developed by Griwodz et. al. with a custom JSON based file format for camera parameters.
Reality Capture (Epic Games, Inc., 2010)		A 3D reconstruction software developed by Epic Games, Inc. Based on XMP (ISO, 2019).
Unity (Unity Technologies, 2022)		A free to use multi-platform game engine developed by Epic Games, Inc. The scene files are based on YAML (Ben-Kiki et al., 2009)
MPEG OMAF (Hannuksela and Wang, 2021)		The <i>Moving Picture Experts Group</i> developed the <i>Omnidirectional Media Format</i> (OMAF) in the standard <i>MPEG-I</i> to facilitate easy interoperability between different devices, components and systems. Uses a custom JSON format.
Local Light Field Fusion (Mildenhall et al., 2019)		A deep learning solution for rendering novel views from input images developed by Mildenhall et al with a custom format based on numpy. Uses a simple numpy format.
NeRF (Mildenhall et al., 2020)		<i>Neural Radiance Fields</i> (NeRFs) are a novel way of reconstructing scenes from input images using neural networks pioneered by Milddenhall et al. Uses a custom JSON format.

Table 1. Supported formats in camorph. **u** marks the default cameras up vector, while **f** marks the front vector.

and objects may appear mirrored in other software packages. The next sections will cover extrinsic parameters and their respective conversions, as well as exemplary intrinsic parameters.

### 3.1 EXTRINSIC CAMERA PARAMETERS

Translations can be represented as a vector addition by the vector  $t$ , which makes them the simplest form of transformations. Rotations on the other hand are linear transformations, and can be represented as  $3 \times 3$  matrices when projecting from  $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ . There is one matrix for counterclockwise rotation around each basis axis when using the standard basis (Vince, 2011). Any arbitrary rotation can be expressed through a combination of three rotations around the angles  $\theta$ ,  $\phi$  and  $\psi$  (Euler, 1775). Rotations can be combined as they are linear transformations by multiplying the matrices.

As matrix multiplication is generally not commutative, the order of rotational operations matter. There are two conventions when applying rotational order: proper Euler angles and Tait-Bryan angles (Markley and Crassidis, 2014). The proper Euler angle representation shares the first axis of rotation with the last, while Tait-Bryan angles rotate around distinct axes. Unfortunately, the terminology is not always consistent. When referring to “Euler angles”, most applications and literature have Tait-Bryan angles in mind. Furthermore, a distinction is made between intrinsic and extrinsic rotations. Intrinsic rotations change the orientation of the associated basis vectors, so that consecutive rotations are in reference to already rotated basis vectors. Extrinsic rotations on the other hand always rotate around the fixed basis axes. Rotation in a 3D system can therefore be represented either by referencing the composite rotational matrix directly, or by supplying three angles and an order by which they should be combined. This is also known as yaw, pitch and roll, where roll can be defined as the rotation about the  $z$ -axis, yaw about the  $y$ -axis and pitch about the  $x$ -axis. However, great care should be taken with this representation, as this definition is not consistent among literature. This means the axes which are described by yaw, pitch and roll can vary (Vince, 2011). Quaternions are another elegant way to represent rotations in 3D space. Originating from trying to generalize complex numbers to 3D space, the rotational and vectorial properties became useful in physics and geometry (Vince, 2021). Unfortunately, there are currently two conventions for displaying quaternions: the Hamilton standard as discussed above, and the JPL standard popularized by the Jet Propulsion Laboratory of NASA (Solà, 2015).

### 3.2 CONVERSION OF EXTRINSIC CAMERA PARAMETERS

A linear transformation can be written as a matrix where the rows are the images of the standard basis vectors. Therefore, to convert from one coordinate system into another, this transformation can be written as a matrix of the images of the basis vectors in the new coordinate system:

$$M_b = \begin{pmatrix} x_i^T \\ y_i^T \\ z_i^T \end{pmatrix} \quad (1)$$

where  $x_i$ ,  $y_i$  and  $z_i$  are the respective image vectors of the  $x$ ,  $y$  and  $z$  basis vectors in the source coordinate system.

The determinant for an orthogonal matrix is always 1 or -1. When the determinant is -1, the matrix represents a combined rotation and reflection (Strampp and Janssen, 2020). With Cartesian coordinates, the only instance where the determinant of  $M_b$  is negative is when converting from a right-handed coordinate system into a left-handed one or vice versa. To split up this conversion, simply multiply  $M_b$  with a reflection matrix along an arbitrary axis.

With  $M_b$ , the camera translation  $t$  in a coordinate system with the basis vectors  $x$ ,  $y$  and  $z$  can be converted to an equivalent translation  $t'$  in another coordinate system with basis vectors  $x_i$ ,  $y_i$  and  $z_i$ :

$$t' = M_b \cdot t \quad (2)$$

When converting a rotation  $R$  however, two extra factors need to be considered:

- Difference in specifying the default orientation of cameras
- $R$  is a linear transformation in the source coordinate system convention, unlike  $t$  which is an offset vector.

The first problem can be solved by finding the rotational difference  $M_d$  between the two standard camera orientations  $M_1$ ,  $M_2$ :

$$M_d = M_1 \cdot M_2^T \quad (3)$$

After this step, the source rotation  $R$  needs to be applied. As a final step, this orientation is converted to the target coordinate system by applying  $M_b$ :

$$M_b \cdot R \cdot M_d \quad (4)$$

This results in these two final core equations for the converted translation  $t'$  and rotation  $R'$ :

$$t' = M_b \cdot t \quad (5)$$

$$R' = M_b \cdot R \cdot M_d \quad (6)$$

### 3.3 INTRINSIC CAMERA PARAMETERS

Cameras are used to project the 3D scene onto a 2D image. The parameters used to define this projection are called intrinsic parameters (Yan et al., 2016). The simplest form is orthographic projection. Objects are projected in parallel lines onto a projection surface. Sizes and angles of objects remain independent of the distance to the camera (Nischwitz et al., 2019).

The most widely used standard camera model with perspective projection is the pinhole camera model. The pinhole is then called center of projection (Tomasi, 2015). The focal length is the distance between the image plane and the center of projection. This is commonly expressed as real-world measurements (mm) or in pixels relative to the dimensions of the output image. The image sensor is a device which collects and samples light and converts it into an electrical signal. The sensor is an rectangular grid of photo-sensitive elements, the size of which can be given in real-world measurements (mm) or as the image resolution in pixels. The most commonly used sensor size in virtual cameras is 36 mm  $\times$  24 mm, which originates in the 135 film photography defined in the ISO standard 1007 (ISO, 2000). Additionally, focal length and sensor size define a field of view. The point where the optical axis pierces the image plane is called the principal point. For a typical virtual camera,

this is in the middle of the image plane. But when using tilt shift where the image plane is moved and rotated in respect to the camera lens, the principal point can vary (Hanning, 2011). The principal point can be expressed in relative or absolute values; either in real-world measurements or in pixels.

#### 4. CAMORPH

*Camorph* is a toolkit to convert different camera parameter representations into each other implemented as a python library, and also supports usage as a command line interface. The core features of *camorph* include:

- Fast and easy conversion of different camera parameter conventions
- Extensibility through a plug-in mechanism
- Handling of crucial properties for different formats
- Automatic computation of different representations of intrinsic camera parameters
- Visualization of camera poses
- Correct computation of intrinsic parameters given a scale factor and cropping coordinates

##### 4.1 PLUG-IN MECHANISM

The system provides an external plug-in mechanism to easily develop additional support for other file formats. The abstract class `FileHandler` defines all necessary methods and properties used by *camorph*. When implementing a plug-in for a new file format, a child class of `FileHandler` has to be created in a subpackage of the package `ext`. Other utility classes can also be placed inside of the subpackage. *Camorph* iterates the packages and loads all instances discovered this way at initialization.

##### 4.2 CRUCIAL PROPERTIES

Some formats need properties other formats do not provide. For example, all photogrammetry formats need source images, while 3D computer graphics formats typically do not support this property because computer graphics typically produces images from virtual scenes while photogrammetry produces a virtual scene from photographs. To alleviate this problem, each plug-in can define a variable `crucial_properties: list[(str, type)]` where crucial properties can be supplied as a tuple of the name and the type. When trying to write to an output format, *camorph* will check if the supplied properties are present for each camera. If not, a template configuration file `config.json` will be written to the output path and an exception is raised. The `config.json` has the following structure:

- `global` holds global information which is applied to all cameras. All properties declared in this scope are set for all cameras. For example, `"global_template": "Value"` will add a property `global_template` with the value `Value` to all cameras when processed. If a property has the suffix `_path`, it is interpreted differently: *camorph* will expect an object with a property `path` and an optional property `filter` as its value. `path` is the

path to a folder as a string, and `filter` is an optional regular expression to filter the files in that folder. For example, `source_image_path` will look for files in the `path` property, filter them according to `filter`, and will then try to match the filenames with the camera IDs. If that is not possible, *camorph* will assume the order of the files in the directory as `os.listdir().sort()` provides is the same order as the cameras in *camorph*. Then, each camera will be assigned a file path as a string according to the ID or order.

- `values` holds local information which is applied to each camera individually. The property `id` is a unique identifier, which can be used to identify the cameras when supplying crucial properties. Local properties will override global properties.

*camorph* will look for a file named `config.json` in the output folder and use this as the configuration file if available.

##### 4.3 AUTOMATIC COMPUTATION OF CAMERA PARAMETERS

Some intrinsic camera parameters have equivalent representations. For example, the focal length can be expressed in millimeters as well as in pixels, the focal length can be expressed as a field of view with a corresponding sensor size, the lens shift can be represented as relative offset to the center of the image plane or as absolute pixel coordinates, and many more. Because of this, some camera formats may have all the information necessary to compute certain parameters. Doing this by hand requires the developer of a `FileHandler` to be aware of the issue, and additionally know all conversion mechanisms. To automate this process, *camorph* will automatically check if enough information is present, and then calculate missing parameters if the information suffices.

Similarly, rotation can be expressed in various ways. When reading any supported format, *camorph* converts the given rotation to quaternions. The resulting rotation and the translation are then converted to *camorphs* internal coordinate system, which is  $z$  up,  $y$  right and  $x$  front with a default camera up vector facing  $y$  and the default front vector facing  $-z$ . When writing to any format, the rotation and translation are converted to the respective target coordinate system as well as to desired representation of extrinsic parameters, for example Euler angles or matrices for rotation.

##### 4.4 SCALING AND CROPPING

When scaling and cropping images, the intrinsic parameters of cameras can change. For example, *FBX* assumes a sensor size of 36 mm × 24 mm by default, regardless of the image dimension. This means that the given focal length in real-world measurements changes when cropping images, as the assumed sensor size remains the same. Similarly, the location of the principal point changes when not retaining the center of the image. These scaling and cropping parameters can be supplied to *camorph*, and the intrinsic parameters of the cameras are scaled accordingly.

##### 4.5 VISUALIZATION

The system provides visualization for orientation and location of cameras in a list. This is done with the python package `matplotlib` (Hunter, 2007), where a red arrow represents the viewing direction (front vector), and a green arrow represents the up vector of a respective camera.

```

from camorph import camorph

cams = camorph.read_cameras('COLMAP', r'\path\to\COLMAP')
camorph.visualize(cams)
camorph.write_cameras('fbx', r'\path\to\fbx\file.fbx', cams)
camorph.write_cameras('unity', r'\path\to\unity\file.unity', cams)

```

Listing 1. Camorph example conversion from COLMAP to FBX and Unity

Format		Avg. memory requirement	Avg. time to write	Avg. time to read
<i>FBX</i>		1415 bytes	1.338 ms	1.256 ms
<i>COLMAP</i> txt	extrinsic	258 bytes	0.997 ms	1.187 ms
	intrinsic	62 bytes		
<i>COLMAP</i> bin	extrinsic	202 bytes	1.104 ms	1.089 ms
	intrinsic	48 bytes		
<i>Meshroom SfM</i>		1722 bytes	0.816 ms	1.012 ms
<i>Reality Capture XMP</i>		1273 bytes	6.293 ms	2.534 ms
<i>Unity scene</i>		2181 bytes	8.359 ms	13.481 ms
<i>MPEG OMAF JSON</i>		1007 bytes	0.866 ms	0.842 ms
<i>LLFF</i>		138 bytes	0.826 ms	0.842 ms
<i>NeRF JSON</i>		1150 bytes	0.895 ms	2.379 ms

Table 2. Memory requirement and time complexity of different formats. The values are given for one camera.

## 5. RESULTS

The performance regarding space and time complexity were analyzed using a workstation with an Intel Xeon E5-2640 v4 CPU with 10 cores at 2.4 GHz. Listing 1 gives an example of how *camorph* can be used to convert cameras saved as *COLMAP* files to an *FBX* or an *Unity* file. Table 3 shows exemplary virtual cameras in *camorphs* visualization and the same set of cameras in *COLMAPS* graphical user interface.

### 5.1 TIME COMPLEXITY

Formats were read and written at an increment of 25 cameras per step up to 300 cameras. To obtain stable results, the conversion was done 100 times and the results were averaged. Table 2 shows that all formats except *Unity* and *Reality Captures XMP* perform similarly. *Reality Captures XMP* has the drawback of having one file for each camera, which creates a lot of read and write overhead. The poor performance of *Unity* however, is due to the performance of *PyYaml* (Simonov, 2016), which is used to parse and create *YAML* files. As most formats do not store the same amount of intrinsic parameters, other comparisons are difficult to draw. For example, reading *FBX* files takes longer than reading simple formats like *LLFF*, but this is also due to the fact that *FBX* stores more parameters per camera and is more flexible in general.

### 5.2 MEMORY REQUIREMENTS

Fig. 2 shows the average memory required to store a camera in different formats. As camera properties can vary (for example, different distortion models have different amount of parameters), cameras in various formats were saved and then averaged.

Similarly to the time complexity, more storage also means more parameters. Formats like *FBX* do have more overhead than the simple *COLMAP* format, but also provide greater flexibility.

## 6. CONCLUSION

The in this paper presented python library *camorph* provides an easy way with minimal external dependencies to convert different representations of camera formats into each other. Additionally, cameras can be visualized in the *camorph* coordinate system. If a multi-software processing (photogrammetry or others) pipeline is used, *camorph* can easily be integrated as a python library to automate conversions. Furthermore, it can be used to verify camera parameters by visualization or by converting them into another easily verifiable format like *FBX*.

As *camorph* is designed to be easily extendable, more formats can be added in the future. Some interesting formats would be the universal scene description (USD) format by Pixar (Sony Pictures Imageworks, 2021), the alembic format (Pixar Animation Studios, 2021) and the Immersive Technologies Media Format (ITMF) (Immersive Digital Experiences Alliance, 2020a)(Immersive Digital Experiences Alliance, 2020b)(Immersive Digital Experiences Alliance, 2021). Additionally, support for formats of *Unreal Engine* (Epic Games, Inc., 2022) as another game engine format would be reasonable.

A graphical user interface may also be useful for *camorph* to increase interactivity. Some features could include selection and manipulation of specific cameras, or saving and converting just a subset of cameras.

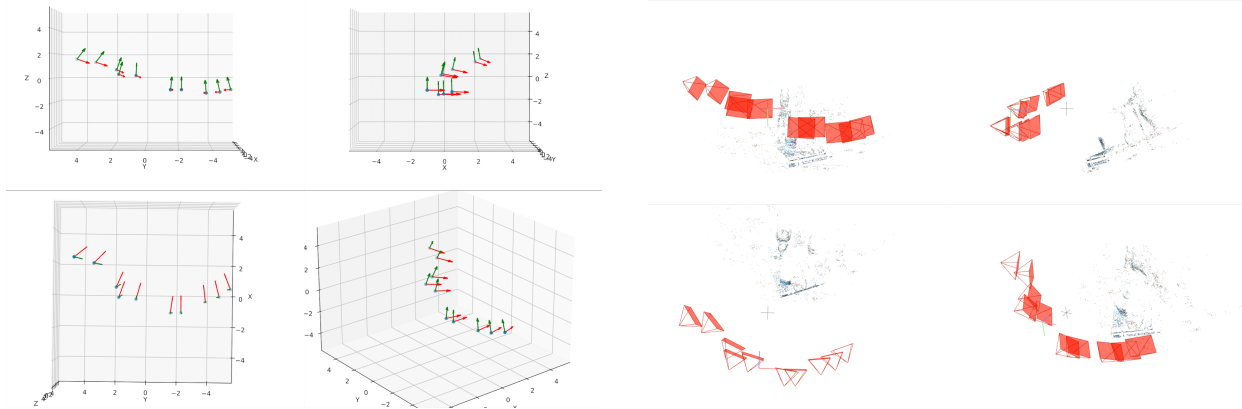


Table 3. Examples of camera parameter representations converted with *camorph*. On the left side is *camorphs* visualization, on the right side the same cameras in the COLMAP coordinate system

Creation of cameras from animation curves could also be implemented. This way, camera arrays could be represented by a single camera and a change of position over time.

## REFERENCES

- AirGon Support, 2018. Converting focal length from pixels to millimeters to use in bentley context capture. <https://support.geocue.com/converting-focal-length-from-pixels-to-millimeters-to-use-in-bentley-context-capture/>. Accessed: 04-11-2022.
- Autodesk, Inc., 2016. Help: Fbx sdk 2016. Accessed: 04-11-2022.
- Autodesk, Inc., 2022. Fbx sdk reference. [https://help.autodesk.com/cloudhelp/2018/ENU/FBX-Developer-Help/cpp\\_ref/index.html](https://help.autodesk.com/cloudhelp/2018/ENU/FBX-Developer-Help/cpp_ref/index.html). Accessed: 04-11-2022.
- Ben-Kiki, O., Evans, C., dot Net, I., 2009. Yaml™ specification index. <https://yaml.org/spec/>. Accessed: 04-11-2022.
- Blender Foundation, 2018. Fbx binary file format specification. <https://code.blender.org/2013/08/fbx-binary-file-format-specification/>. Accessed: 04-11-2022.
- Blender Foundation, 2022. Blender. <https://github.com/blender>.
- Boyce, J. M., Dore, R., Dziembowski, A., Fleureau, J., Jung, J., Kroon, B., Salahieh, B., Vadakital, V. K. M., Yu, L., 2021. MPEG Immersive Video Coding Standard. *Proceedings of the IEEE*, 1–16. <https://ieeexplore.ieee.org/document/9374648/>.
- Cernea, D., 2020. OpenMVS: Multi-view stereo reconstruction library.
- Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H., 2022. TensorRF: Tensorial Radiance Fields. *arXiv preprint arXiv:2203.09517*.
- Doré, R., Kroon, B., Hannuksela, M., Champel, M.-L., Boyce, J., 2018. Call for test materials for 3dof+ visual. Standard, International Organisation for Standardisation, Coding of Moving Pictures and Audio.
- Epic Games, Inc., 2010. Reality Capture. <https://www.capturingreality.com/>.
- Epic Games, Inc., 2022. Unreal engine. Accessed: 04-11-2022.
- Euler, L., 1775. *Formulae generales pro translatione quacunque corporum rigidorum*. *Novi Commentarii Academiae Scientiarum Imperialis Petropolitanae*.
- Griwodz, C., Gasparini, S., Calvet, L., Gurdjos, P., Castan, F., Maujean, B., Lillo, G. D., Lanthony, Y., 2021. Alicevision Meshroom: An open-source 3D reconstruction pipeline. *Proceedings of the 12th ACM Multimedia Systems Conference - MMSys '21*, ACM Press.
- Hanning, T., 2011. *High Precision Camera Calibration*.
- Hannuksela, M. M., Wang, Y.-K., 2021. An Overview of Omnidirectional Media Format (OMAF). *Proceedings of the IEEE*, 109(9), 1590-1606.
- Hunter, J. D., 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.
- Immersive Digital Experiences Alliance, 2020a. Itmf data encoding specification. Technical report.
- Immersive Digital Experiences Alliance, 2020b. Itmf scene graph specification. Technical report.
- Immersive Digital Experiences Alliance, 2020c. An overview of scene description formats. Technical report.
- Immersive Digital Experiences Alliance, 2021. Itmf container specification. Technical report.
- ISO, 2000. Photography - 135-size film and magazine - Specifications. Standard, International Organization for Standardization, Geneva, CH.
- ISO, 2019. Graphic technology — Extensible metadata platform (XMP) — Part 1: Data model, serialization and core properties. Standard, International Organization for Standardization, Geneva, CH.
- Markley, L., Crassidis, J., 2014. *Fundamentals of Spacecraft Attitude Determination and Control*.
- Mildenhall, B., Srinivasan, P. P., Ortiz-Cayon, R., Kalantari, N. K., Ramamoorthi, R., Ng, R., Kar, A., 2019. Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines. *ACM Transactions on Graphics (TOG)*.



- Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., Ng, R., 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *European Conference on Computer Vision*.
- Müller, T., Evans, A., Schied, C., Keller, A., 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics (TOG)*, 41(4), 102:1–102:15. <https://doi.org/10.1145/3528223.3530127>.
- NAVER LABS Corp., 2021. Kapture. <https://github.com/naver/kapture>. Accessed: 04-11-2022.
- Nischwitz, A., Fischer, M., Haberäcker, P., Socher, G., 2019. *Computergrafik*. Springer Vieweg.
- Pixar Animation Studios, 2021. Alembic. <https://github.com/alembic/alembic>. Accessed: 04-11-2022.
- Reality Capture Support, 2022. Reality Capture XMP Camera Math. <https://support.capturingreality.com/hc/en-us/articles/360017783459-RealityCapture-XMP-Camera-Math>. Accessed: 04-11-2022.
- Schönberger, J. L., 2020. Colmap. <https://colmap.github.io/>. Accessed: 04-11-2022.
- Schönberger, J. L., Frahm, J.-M., 2016. Structure-from-Motion Revisited. *Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Schönberger, J. L., Zheng, E., Pollefeys, M., Frahm, J.-M., 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. *European Conference on Computer Vision (ECCV)*, Amsterdam, The Netherlands.
- Side Effects Software Inc., 2022. Sidefx: Houdini. Accessed: 04-11-2022.
- Simonov, K., 2016. Pyyaml. <https://pyyaml.org/>. Accessed: 04-11-2022.
- Solà, J., 2015. Quaternion kinematics for the error-state KF.
- Sony Pictures Imageworks, 2021. Universal scene description. <https://graphics.pixar.com/usd/release/index.html>. Accessed: 04-11-2022.
- Strampp, W., Janssen, D., 2020. *Höhere Mathematik 1 Lineare Algebra*. Springer Vieweg.
- The Khronos® 3D Formats Working Group, 2021. glTF™ 2.0 specification. <https://www.khronos.org/registry/glTF/specs/2.0/glTF-2.0.html>. Accessed: 04-11-2022.
- Tomasi, C., 2015. A simple camera model.
- Unity Technologies, 2022. Unity - manual: Unity manual. <http://docs.unity3d.com/Manual/index.html>. Accessed: 04-11-2022.
- Vince, J., 2011. *Rotation Transforms for Computer Graphics*. Springer-Verlag London.
- Vince, J., 2021. *Quaternions for Computer Graphics*. Springer-Verlag London Ltd.
- Yan, K., Tian, H., Liu, E., Zhao, R., Hong, Y., Zuo, D., 2016. A Decoupled Calibration Method for Camera Intrinsic Parameters and Distortion Coefficients. *Mathematical Problems in Engineering*, 2016, 1-12.
- Zhang, Z., 2000. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22.