DTU Library

The Ultrasound File Format (UFF) - First draft

Bernard, Olivier; Bradway, David; Hansen, Hendrik H.G.; Kruizinga, Pieter; Nair, Arun; Perdios, Dimitris; Ricci, Stefano; Rindal, Ole Marius Hoel; Rodriguez-Molares, Alfonso; Stuart, Matthias Bo; Dos Santos, Pedro Filipe Viseu

Published in:
Proceedings of 2018 IEEE International Ultrasonics Symposium

Link to article, DOI:
10.1109/ULTSYM.2018.8579642

Publication date:
2018

Document Version
Peer reviewed version

The Ultrasound File Format (UFF) - First draft

Olivier Bernard *
CREATIS
University of Lyon,
Lyon, France

David Bradway
Dept. of Biomedical Engineering
Duke University
Durham, United States

Hendrik H.G. Hansen
Medical UltraSound Imaging Center
Radboud University Medical Center
Nijmegen, Netherlands

Pieter Kruizinga
Biomedical Engineering
Erasmus MC
Rotterdam, Netherlands

Arun Nair
PULSE Lab
Johns Hopkins University
Baltimore, United States

Dimitris Perdios
Signal Processing Laboratory
EPFL
Lausanne, Switzerland

Stefano Ricci
Engineering Information Dept.
University of Florence
Florence, Italy

Ole Marius Hoel Rindal
Dept. of Informatics
University of Oslo
Oslo, Norway

Alfonso Rodriguez-Molares
Dept. Circulation and Medical Imaging
NTNU
Trondheim, Norway

Matthias Bo Stuart
Biomedical Engineering
Technical University of Denmark
Lyngby, Denmark

Pedro Filipe Viseu Dos Santos
Dept. of Cardiovascular Imaging and Dynamics
KU Leuven
Leuven, Belgium

***Abstract*—The lack of a standard format for storing ultrasound research data is hindering our ability to share and compare research results. This is slowing down the progress in our field, making it difficult to assess the relevance of new techniques.**

In October 2017 the Ultrasound File Format (UFF) initiative was formed with the aim of defining such a standard, with the support of eleven research groups. Here we present some of the components of the first draft of the UFF format.

***Index Terms*—ultrasound, file, format, channel data, data sharing, research data, data reuse**

I. Introduction

Systems able to produce channel data have become widely available to research laboratories, enabling the development of advanced sequences and processing techniques.

Despite its popularity, channel data are often stored in formats defined by the laboratories themselves, meeting their particular interests and tools. Those formats are often highly efficient, but they are fitted to specific sequences, systems, or probes, omitting information that would be required by a third party to use the data. They are difficult to maintain and expand.

Due to the increasing complexity of ultrasound imaging techniques, it no longer suffices with a set of images to assess the validity, correctness, and relevance of new techniques [1]. In recent years, there has been an increase in willingness to embrace data sharing practices [2]. Over 63% of researchers submit research data files as supplementary information for their manuscripts, or deposit the files in an open repository. It has been shown [2] that this practice increases the return on investment, enabling the reproduction, assessment, and reuse of research.

However, in the field of ultrasound imaging, the prevalence of custom formats hampers the diffusion of data.

In the occasion of the IEEE IUS 2017 conference (Washington D.C), the Ultrasound File Format (UFF) initiative was started. Eleven laboratories agreed upon the need for a common format for channel data that would facilitate dissemination of data, replication of results, and comparison of processing techniques.

And hence the UFF taskforce was formed, composed by one member from each laboratory, with the objective of developing the common format. During most of 2018 the UFF taskforce has worked in the first draft of the standard. A draft that is disclosed in the following sections.

II. Methods

All units are given in SI units unless otherwise specified. We use blue color to identify *references* to UFF objects, and green to identify *arrays* of UFF objects.

Hierarchical Data Format (HDF5) was chosen due to its ability to store and organize large amounts of data [3]. HDF5 is a platform independent data model and file format that supports an unlimited variety of datatypes. It is designed for flexible and efficient I/O in local and remote systems.

HDF5 is developed by the HDF Group, a non-profit organization with the mission of advancing state-of-the-art open source data management technologies. HDF5 has become a *de facto* standard in the scientific community. Many scientific environments count with a HDF5 API such as MATLAB [4], Python [5], or Julia [6].

HDF5 uses two objects types: *groups* and *datasets*. HDF5 groups organize data objects similarly to directories and files in UNIX. Every HDF5 file contains a root group that can contain other groups. HDF5 datasets contain data and metadata that describe the data [3].

We use the term *UFF object* to refer to an independent set of data. UFF objects are stored as HDF5 groups, and are suited to be implemented as classes in several programming languages.

*Authors are shown in alphabetical order.

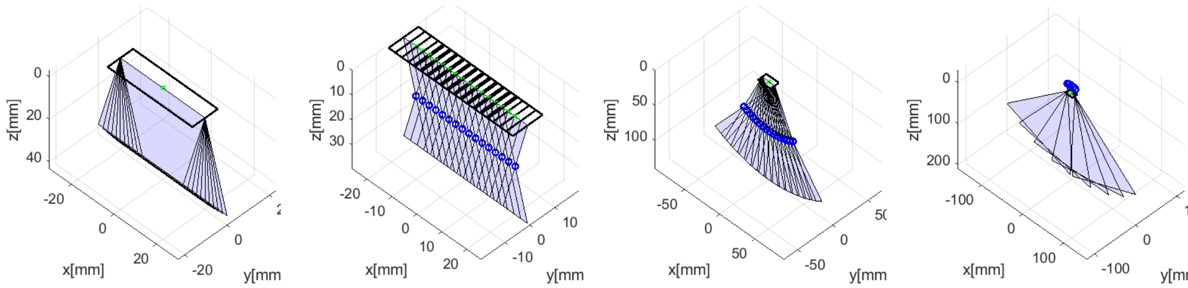


Fig. 1: Example of sequences

III. Results

In this section we describe the objects that have been included in the first draft of the UFF.

A. Channel data

The object `uff.channel_data` contains all the information needed to store and later process channel data,

```
uff.channel_data
 ├─ authors < string
 ├─ description < string
 ├─ local_time < string ISO 8601
 ├─ country_code < string ISO 3166-1
 ├─ system < string
 ├─ sound_speed < double
 ├─ repetition_rate < double
 ├─ probes < uff.probe
 ├─ unique_waves < uff.wave
 ├─ unique_events < uff.event
 ├─ sequence < uff.timed_event
 │   ├─ event < uff.event
 │   └─ time_offset < double
 └─ data < double
```

where `authors` identify the authors of the data; `description` describes the acquisition scheme, motivation and application; `local_time` and `country_code` identify the time and place the data were acquired; `system` describes the hardware used in the acquisition; `sound_speed` contains the reference speed of sound that was used in the system to produce the transmitted waves; and `repetition_rate` is the sequence repetition rate, also referred to as framerate in some scenarios

The object `uff.channel_data` contains all the `probes` used in the acquisition, a list of the `unique_waves` that have been transmitted, and a list of the `unique_events` that form the sequence. The `sequence` is specified as an array of `uff.timed_events`, each member containing an `event` reference, and the `time_offset` since the beginning of the beginning of the current repetition, also referred to as frame.

The HDF5 dataset `data` contains the channel data, organized as a matrix of four dimensions:

```
samples × channels × events × repetitions
```

where `samples` is the number of temporal samples acquired by the system, `channels` is the number of active channels, `events` is the number of events in the sequence (not unique events), and `repetitions` is the number of times the sequence was repeated.

This proposal has the limitation of requiring that all event acquisitions have the same number of time samples and active channels.

B. Events

We define `uff.event` as a tuple of `uff.transmit_setup` and `uff.receive_setup` objects,

```
uff.event
 ├─ transmit_setup < uff.transmit_setup
 │   ├─ probe < uff.probe
 │   ├─ transmit_waves < uff.transmit_wave
 │   │   ├─ wave < uff.wave
 │   │   ├─ time_offset < double
 │   │   └─ weight < double
 │   └─ channel_mapping < int32
 └─ receive_setup < uff.receive_setup
     ├─ probe < uff.probe
     ├─ time_offset < double
     ├─ sampling_frequency < double
     └─ channel_mapping < int32
```

The object `transmit_setup` contains a reference to the `probe` used to transmit the waves, an array describing all the `transmit_waves`, and a `channel_mapping` dataset routing the elements in the probe to the active channels in the system.

Each of the `transmit_waves` contains a reference to a `wave`, and the `time_offset` since the start of the event. Note that several waves can be transmitted simultaneously, with different geometry, `weight` and `time_offset` relative to each other.

The object `receive_setup` contains a reference to the `probe` used on receive, the `time_offset` between the event start and the beginning of data acquisition, the `sampling_frequency`, and a `channel_mapping` dataset routing the elements in the probe to the channels in the system.

C. Waves

The geometry of an ultrasonic wave is defined as

```
uff.wave
 |__origin < uff.transform
 |__wave_type < enumeration
 |__aperture < uff.aperture
 |   |__origin < uff.position
 |   |__window < string
 |   |__f_number < double
 |   |__fixed_size < double
 |__excitation < uff.excitation
```

where `wave_type` is an enumerated type (int32) that indicates the type of transmitted wave: converging wave (0), diverging wave (1), plane wave (2), or cylindrical wave (3). The interpretation of `origin` depends on `wave_type`. For converging waves `origin.translation` holds the location of the wave focal point, for diverging waves `origin.translation` holds the location of the virtual source; in both cases the `origin.rotation` is ignored. For plane waves `origin.rotation` holds the orientation of the wave and `origin.translation` is ignored.

While `origin` completely defines the transmit delay profile, the `aperture` defines the transmit apodization profile. In this case `aperture.origin` defines the center of the aperture. The size of the aperture can be described with `fixed_size`, or with `f_number` in which case the aperture size is $A = d/F$ where F is the `f_number` and d is the distance between `uff.wave_geometry.origin` and `uff.wave_geometry.aperture.origin`.

The object `window` is a string describing the apodization window. Examples of `uff.wave` are shown in Fig. 2.

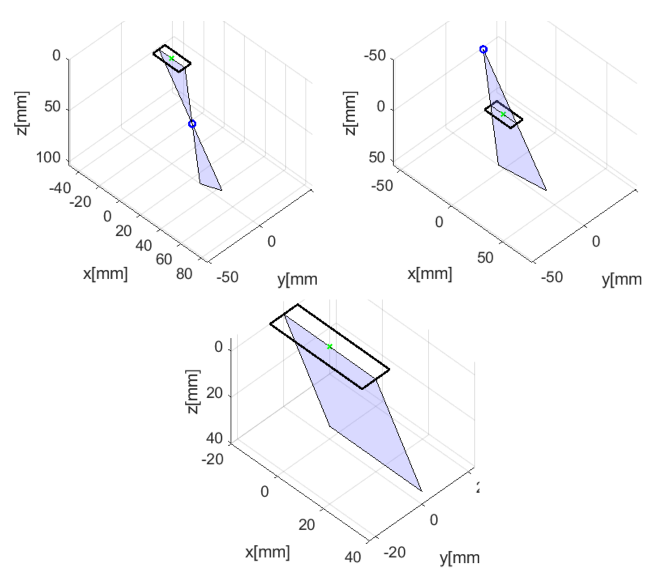


Fig. 2: Example of waves defined with `uff.wave`

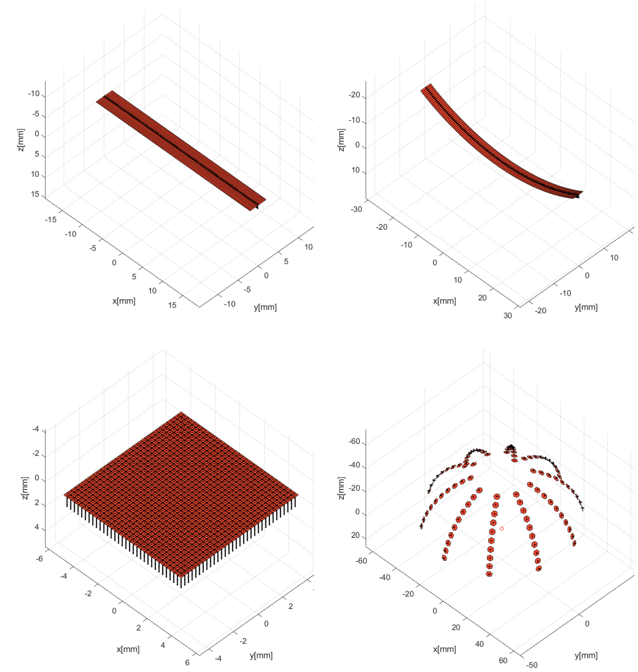


Fig. 3: Example of probes defined with `uff.probe`

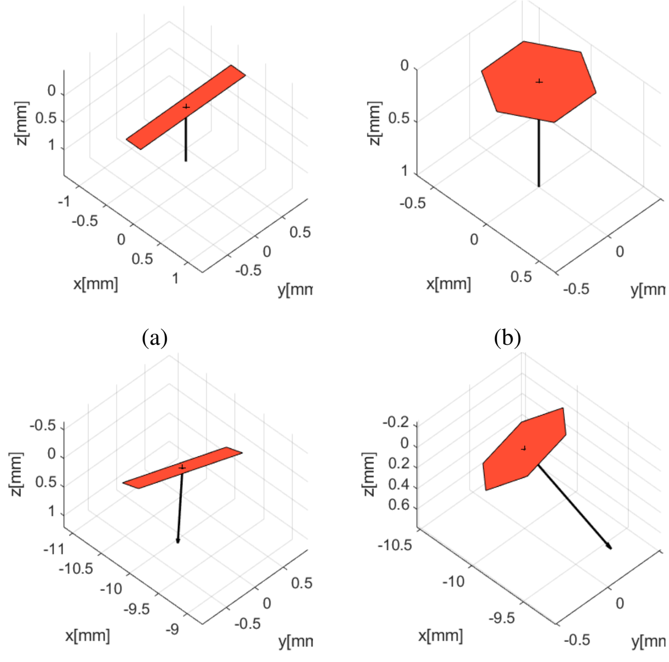
D. Probes

UFF aims to support all possible probe geometries by defining `uff.probe` as an arbitrary collection of elements

```
uff.probe
 |__transform < uff.transform
 |__focal_length < double
 |__elements < uff.element
 |   |__transform < uff.transform
 |   |__geometry < uff.element_geometry
 |   |__impulse_response < uff.impulse_res...
 |__element_geometries < uff.element_geom...
 |   |__perimeter < uff.perimeter
 |__impulse_responses < uff.impulse_response
     |__initial_time < double
     |__sampling_frequency < double
     |__data < double
     |__units < string
```

where `focal_length` specifies the lens focusing distance. Note that the elements in `element_geometry` and `impulse_response` are referred by the fields `geometry` and `impulse_response` of each member in `element`. This avoids unnecessary replication of information.

The objects in `elements` describe an ultrasonic element with a given geometry and impulse response, located at a given location in space. The objects in `element_geometry` define the geometry of the elements that have unique geometry (i.e. in a linear array `element_geometry` will have size 1).



(a) (b) (c) (d)

Fig. 4: Example of element geometries (a and b) and corresponding elements after transformation (c and d)

Here we assume that the acoustic center of the element is at origin $O = (0, 0, 0)$ pointing towards $\vec{z} = (0, 0, 1)$. The element shape is defined by a closed `perimeter` contained within the XY-plane, that is in turn composed of an ordered set of `uff.position` instances. In Fig. 4 examples are shown of two different element geometries and the attitude they take after applying the transform in the corresponding `element`.

The objects in `impulse_responses` describe the the two-way impulse response of unique elements, where `initial_time` denotes the time offset between the activation of the impulse and the first acquired sample. Note that `geometry` and `impulse_response` are references (*H5Link*).

The `probe` contains a `uff.transform` object that enables moving the probe in space, a convenient feature in probe tracking applications, or if more than one probe is used.

E. Transforms

The object `uff.transform`, which is included in many UFF objects, defines an affine transformation in a 3D Cartesian system, as

```
uff.transform
 ├─ translation < uff.translation
 └─ rotation < uff.rotation
```

A transform is composed by a rotation $R = (r_x, r_y, r_z)$ followed by a translation $T = (t_x, t_y, t_z)$. The rotation axis X, depicted in Fig. 5, is known as *elevation*, the rotation axis Y is known as *azimuth*, and the rotation axis Z is known as *roll*, in aeronautical terminology.

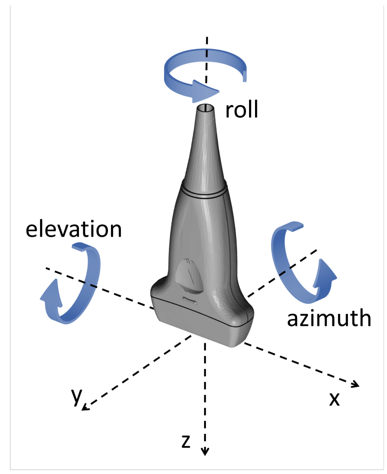


Fig. 5: 3D Cartesian axes and rotations.

IV. Conclusion

We present here the main components in the first draft of the Ultrasound File Format (UFF). The scope of this first draft is limited to the specification of the components needed to store and process channel data. The UFF aims to be as general as possible, covering all possible probes, systems, and transmission schemes.

The format is based on HDF5, a platform independent file format designed for efficient handling of large amounts of data.

UFF is developed by the UFF taskforce, a group of researchers aiming to facilitate and popularize the dissemination of research data, replication of results, and comparison of processing techniques.

At the moment of this publication, the first draft is being discussed and revised by the UFF taskforce. However files and code examples are available at https://bitbucket.org/ultrasound_file_format/uff/wiki/Home.

Acknowledgment

The UFF taskforce would like to thank the IEEE IUS organizing committee for providing a venue for UFF meetings, and our home institutions for supporting our activity in the UFF initiative.

References

[1] O. M. H. Rindal, A. Austeng, H. Torp, S. Holm, and A. Rodriguez-Molares, “The dynamic range of adaptive beamformers,” in *2016 IEEE International Ultrasonics Symposium (IUS)*, pp. 1–4, Sept 2016.

[2] N. A. Vasilevsky, J. Minnier, M. A. Haendel, and R. E. Champieux, “Reproducible and reusable research: are journal data sharing policies meeting the mark?,” *PeerJ*, vol. 5, p. e3208, 2017.

[3] The HDF Group, “Introduction to hdf5.” https://portal.hdfgroup.org/display/HDF5/Introduction+to+HDF5, 2018. [Online; accessed 2018-09-17].

[4] Mathworks, “Hdf5 files.” https://www.mathworks.com/help/matlab/hdf5-files.html?searchHighlight=hdf5&s_tid=doc_srchtitle, 2018. [Online; accessed 2018-09-17].

[5] A. Collette, “Hdf5 for python.” http://docs.h5py.org/en/stable/, 2018. [Online; accessed 2018-09-17].

[6] T. Holy, “Julia hdf5 guide.” https://github.com/JuliaIO/HDF5.jl/blob/master/doc/hdf5.md, 2018. [Online; accessed 2018-09-17].