C3D File Format Extensions

Draft Proposal: 23 April 2001
Author: Jonathan Attias

Vicon Motion Systems Ltd

Assumptions

It is assumed that the reader is familiar with the original C3D file specification. Vicon Motion Systems produce a document describing our use of the format. Further information on the standard format is available at http://www.c3d.org/.

Overview

This document describes a mechanism for extending C3D files in a consistent manner and suggests one extension to deal with the issue of storing measurement data from motion capture systems that use more than the current C3D limit of seven cameras.

C3D files have always been capable of storing arbitrary data sections but, to date, most if not all applications support only the originally defined sections containing parameters and combined motion and analogue data. These sections are special because the C3D header section contains explicit pointers (record number references) to locate these sections.

There are a number of undefined words in the C3D header that could be used as pointers to new data sections. In fact, there are so many (159 according to the original specification) that these should suffice for all future needs. However, the problem with this approach is that the assignment of slots (undefined words) must be agreed by committee. Any organisation should be able to start developing a new data section without having to first obtain an agreed slot number, which could be very time consuming. Picking a slot at random in the hope that it doesn't clash with anybody else's choice is not a very appealing approach.

Apart from the arbitrary nature of the slot choices, every application vendor will need to be aware of everyone else's extensions and understand each slot as it is assigned. In other words, we would have to hard code knowledge that slot 42 corresponds to the "FROZEN PEAS" data section and slot 127 corresponds to the "ELECTRIC MONK" data section, etc. This is important because it is necessary to know exactly where each section starts and ends in order to preserve data that you don't understand when loading and then saving files. For example, there is currently no information in the C3D file to determine where the parameters section ends. All we can assume is that it ends where the next section starts but if we don't know that one or more sections have been inserted between the parameters and 3D data sections (because we haven't kept up to date with the latest agreed slot assignments) then we risk stomping all over those sections with new parameters or truncating them because we don't think the parameters section needs to be that big.

Data section extension proposal

In this proposal, we take advantage of the fact that every C3D file has a parameters section and virtually every application that reads C3D files contains code to read and write these parameters in a standard manner.

The proposal is simply to add a new parameter group called "CONTENTS" that lists the data sections by name and the record numbers where they can be found. An acknowledgment must at this point be made to Andrew Danis since we jointly discussed this idea at a meeting some years ago.

There is still a risk that different vendors or organisations may start to develop sections with the same name but this is thought to be less likely. If the section is essentially private

to a particular organisation then some organisation specific prefix (or suffix) could be included in the section name to be certain of uniqueness.

If the section is generally applicable then it is probably best to involve others in discussion to avoid duplication of effort on the same type of section but that is a different matter from having to globally agree to a slot number before even experimenting with a new data section.

CONTENTS: group

So the “CONTENTS” parameter group would go something like this:

Name	Type	Dimension	Usage
LABELS	ASCII	(L, N)	N section labels of length L (typically 16 characters).
NAMES (?)	ASCII	(L, N)	N human readable section names of length L (typically 16 characters). The LABELS parameters are intended for programmatic use and should never change for a given section. Having a separate NAMES parameter allows for a human friendly, language-specific version to be presented to the user along with a description (see below) to show them the file contents.
DESCRIPTIONS (?)	ASCII	(L, N)	N section descriptions of length L (typically 64 characters). Descriptions are always optional in that they are often left blank. It might be difficult to agree on descriptions for standard sections but they might be useful for unusual sections and can also be language specific to suit local users.
DATA_STARTS	Integer	(N)	N 1-based record numbers corresponding to the start of each data section. Records are 512 bytes in length.
SIZE (?)	Integer	(N)	N values corresponding to the size of each data section in 512-byte records. In principal, every section should be listed in this group so this parameter should not be necessary. The sizes can be calculated by examining the DATA_STARTS value of the following section. Although useful for making it easier to determine the size of a section without having to scan the rest of the contents, it does place some further burden on file writers, especially when it comes to the (usually) final 3D data section where the size might not be known until it is completely written. It is suggested that this parameter not be included but is offered here for discussion purposes.

Although not strictly necessary, the sections should be listed in the group tables in ascending order of START_RECORD just to make life easier for everyone, especially if the SIZE parameter is not included.

Standard sections:

Label	Description	Typical Start Record
PARAMETERS	Blah	2
3D_DATA	Interleaved 3D points and analogue data	10

Related parameters

Many data sections will require a number of parameters to allow them to be correctly interpreted. One option is to embed all of this information into a header of the data section itself. However, there is a perfectly good standard “PARAMETERS” section already in the file which everyone understands and has code to read from and write to. The preference is therefore to use this section wherever possible.

Discussion points

What parameter group should the related parameters belong to? If a section is completely independent then it makes sense for the related parameters to form their own group. Sometimes, the parameters might need to form multiple groups (such as the existing POINT and ANALOG parameter groups which both relate to the same 3D data section). If the section represents an extension to existing data for which a group already exists then there is an argument for adding the new parameters to the existing group.

If I form a new parameter group, what should it be called? The answer to this depends on how many new parameter groups need to be formed to service the new data section. If there is only one then the names should correspond. It makes sense for the parameter group name to exactly match the data section label. The C3D convention thus far tends towards singular naming for parameter groups. So we have POINT rather than POINTS and FORCE_PLATFORM rather than FORCE_PLATFORMS, etc. If the singular convention is followed then the section labels will also tend towards singular variants, which can be a little odd or counter-intuitive at times.

Should underscores be used everywhere or not? A further convention within the C3D file is to use underscore (‘_’) characters to separate words in parameter and group names rather than spaces. There is nothing in the C3D specification that makes this a necessity since the names are always provided with a length field. However, there may be applications that interpret parameters embedded within textual instructions that may use spaces to separate elements. These applications would be unable to distinguish the parameters correctly if they contained spaces. The same applies to data section labels, which should therefore also use underscores rather than spaces.

What about the questionable parameters in the CONTENTS group (or should that be CONTENT group)? Should the NAMES, DESCRIPTIONS and SIZE parameters be included or not?

What about DATA_START parameters? If the CONTENTS group contains the start record for each data section then a separate parameter should not be necessary. Some data sections might not even have a parameter group. However, a POINT:DATA_START parameter already exists (though there is no corresponding ANALOG:DATA_START parameter) and it is a lot more convenient for a reader that knows about a particular data section to get the start record directly from a SECTION:DATA_START parameter, if present, than to scan through the CONTENTS:LABELS parameter list for “SECTION” and then extract the corresponding CONTENTS:DATA_STARTS parameter value. The downside is duplication of storage and the risk of inconsistency between them.

Extended storage of camera masks

The C3D file specification for 3D point data allows for only 7 bits of camera mask information to be associated with each data point, regardless of whether the data is stored in integer or real format. Since the format's introduction many years ago, systems are now widely available with more than 7 cameras.

There have been many ideas in recent times for incorporating the extra information into the 3D data but most, if not all, so far have involved changes that are not 100% backward compatible. They would necessitate that applications be changed in some way in order to cope with these extended files correctly. Also, many of the schemes extended the mask only up to 24 cameras, which is not sufficient for future demands.

One example of a prior approach forced the use of real format data and making use of the extra 16 bits that become available. This only allowed for 23 cameras (not enough by today's standards) and potentially meant storing invalid floating point values. It also meant a re-interpretation of the data so an extra flag would have been necessary which legacy software would simply not know to read.

Another approach involved extending the point data structures themselves with similar but more drastic breaking of existing code.

Requirements

Any scheme for incorporating further camera mask information must satisfy these requirements:

- Any number of cameras
- 100% backward compatibility

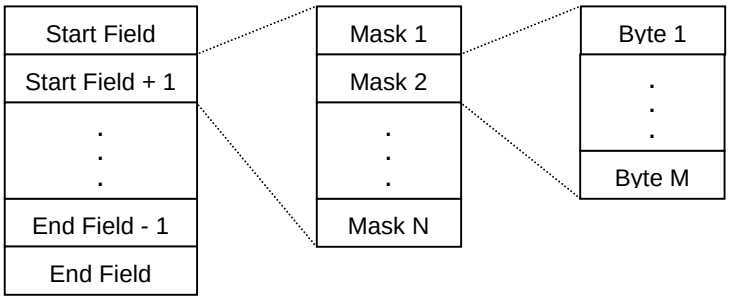
By saying that the scheme must be 100% backward compatible, we mean that we must not have to modify any existing software in order for it to successfully read extended data files. There is too much legacy software around that would be totally impractical to update. Of course, that does not mean that this software will be able to extract the new information, only that the information it already reads will not be affected in any way.

Proposed solution

Given the flavour of this document, the proposed solution not surprisingly involves the introduction of a new data section. It is the only way to guarantee backwards compatibility and satisfy the increasing demands for more cameras.

"CAMERA_MASK" data section

Note that the name follows the singular naming convention for parameter groups. Data appears as one mask for each specified point trajectory for each field as follows:



Where N is the number of point trajectories for which camera masks need to be stored and M is the number of bytes required to store the masks for every camera. The mask bytes are stored least significant first so they correspond to cameras as follows:

Byte 1: cameras 1 - 8

Byte 2: cameras 9 – 16

Byte 3: cameras 17 – 24

Byte 4: cameras 25 – 32

etc.

The total storage required for the section in bytes is:

Total Bytes = F * N * M

Where: F = #fields = End Field – Start Field + 1,
N = #trajectories
M = #bytes per mask = (#cameras – 1) DIV 8 + 1

Note that DIV means integer division.

If the number of cameras (#cameras) is less than 8 (i.e. 2 to 7) then this new section can be omitted entirely since the original scheme is perfectly adequate.

Parameter storage

A few parameters are required to correctly interpret the contents of the CAMERA_MASK data section:

- The number (M) of bytes stored for each camera mask
- The number (N) of trajectories for which masks are stored
- The correspondence of these trajectories with those in the 3D data section

It is questionable whether these parameters should belong to the existing POINT group or form a new parameter group. For cleanliness, it is suggested that a new CAMERA_MASK group be formed. The advantage is that if the CAMERA_MASK data section is not present then the whole CAMERA_MASK parameter group would be omitted rather than just a few parameters from the POINT group. Also, the parameter names are much simpler.

Name	Type	Dimension	Usage
DATA_START	Integer	Scalar	1-based record number corresponding to the start of the CAMERA_MASK data section. Records are 512 bytes in length.
MASK_SIZE	Integer	Scalar	The number of bytes stored for each point camera mask.
USED	Integer	Scalar	The number (N) of trajectories for which extended camera masks are stored.
LABELS	ASCII	(L, N)	N trajectory labels of length L associated, by order, with masks in the CAMERA_MASK data section. The labels must also correspond to those found in the POINT:LABELS[n] lists though they may appear here in a different order and not all trajectories may have extended camera masks.

LABELS2	ASCII	(L, N)	Additional labels beyond the 255 limit in LABELS. Note that LABELS3, LABELS4, etc. parameters may also be present as necessary.

Existing masks

At the time of writing, the Vicon Workstation software wraps camera contributions onto the available 7 bits so bit 8 represents a union (binary OR'ing of) cameras 1, 8, 15 and 22, bit 9 represents cameras 2, 9, 16, and 23, etc. It was thought that, given the inability to correctly hold all of the mask information, it was more useful to indicate 'some' contribution by cameras rather than none at all if not seen by any of the first 7 cameras. Some applications use this information to determine whether a point was measured (has camera contributions) or calculated (has no contributions). With the introduction of new visualisation features, this has turned out to be misleading since the actual contributing camera is ambiguous and can therefore be misinterpreted.

Instead, it is better to store only the first 7 cameras in the existing camera masks so that they are at least correct. The correct way to determine whether a point is measured or calculated is to look at the point residual. If this is zero then the point is calculated, otherwise it is measured.

Usage strategies

1. If reading all point data into memory:
    a. Read in all the data as before with the old camera masks.
    b. Check for the existence of the CAMERA_MASK parameter group and data section.
    c. For each field, read the extended camera masks and patch up the in-memory masks of the appropriate trajectories.

2. If reading a single trajectory point:
    a. Read in the trajectory point and old camera mask as before.
    b. Check for the existence of the CAMERA_MASK parameter group and data section.
    c. Check if the trajectory is included in the extended mask section.
    d. Calculate the mask data index, extract the extended mask and patch up the point data.

3. When writing a C3D file:
    a. Determine whether the extended camera masks are necessary or not (are there more than 7 cameras?)
    b. Determine which trajectories need extended mask storage. If this cannot easily be determined, assume all trajectories.
    c. Set up the CAMERA_MASK parameters and write the parameters section as normal.
    d. Write other data sections as appropriate.
    e. Write the CAMERA_MASK data section.
    f. Write the 3D data section as before.