File Format Specification MMPLD

Version: 1.2 Release
Author: Sebastian Grottel
Date: 17.05.2016

Preface

The file formats MMPLD and MMDPLD basically are binary memory dumps of MegaMol's internal data structures, namely the content of the `MultiParticleDataCall`. The corresponding data files can be loaded using the Modules `MMPLDDataSource`. To generate these files from any compatible data source you can use a MegaMol job based on the `DataWriterJob` and the Modules `MMPLDWriter`. See Appending A for an example project file.

This file format is intended for visualization using MegaMol for simple, small and medium sized data sets (max. about 1 million particles and less than thousand configurations).

Data in this file formats is stored as *little-endian* binary data. The term `uint32` corresponds to a 32 bit large unsigned integer value; `uint16` and `uint64` are defined accordingly. The term `byte` corresponds to an 8 bit large unsigned integer value. The term `float` corresponds to a 32 bit large floating point value following the IEEE standard[1] (cf. `real` or `single`). The term `double` corresponds to a 64 bit large floating point value following the IEEE standard.

MMPLD – Version 1.2

Header Data

The file starts with 60 bytes header data:

Bytes	Data	Description
0..5	char*	Magic identifier
6..7	unsigned int 16 bit	Version number
8..11	unsigned int 32 bit	Number of data frames
12..35	6x float (32 bit)	Data set bounding box
36..59	6x float (32 bit)	Data set clipping box

The Magic identifier must have the value „MMPLD\x00" as ASCII: `4D 4D 50 4C 44 00`.

The version number encodes the major version as multiple of 100 and adds the minor version.

```
ver =  maj_ver * 100 + min_ver
maj_ver = ver / 100
min_ver = ver % 100
```

The version number must be `102` for this file format version.

[1] IEEE Standard for Binary Floating-Point Arithmetic for microprocessor systems (ANSI/IEEE Std 754-1985)

The number of data frames must be at least 1. The remaining text will refer to this value as *frame count*.

The data set bounding box is stored as six float values: minimum X value, minimum Y value, minimum Z value, maximum X value, maximum Y value, and maximum Z value. The differences between minimum and maximum value on each axis must not be zero or negative.

The data set clipping box is stored in the same way as the bounding box.

The bounding box specifies the reference bounding box of the data set, e.g. the simulation box size. The clipping box specifies the extent of the visual representations of the data set. For example, if the particle positions are always inside the bounding box, but the particles are represented as spheres with different radii, then the clipping box should be the bounding box inflated in all directions by the maximum sphere radius.

Seek Table

The seek table is stored directly following the header data. For each time frame one `unsigned int 64 bit` value is written. These values are the byte offsets from the beginning of the file to the beginning of the data of the corresponding time frame data. A value of zero (or less than 60) should be treated as error. An additional last `unsigned int 64 bit` value written specifies the byte offset from the beginning of the file to the end of the data of the last time frame. This value should be the file size itself. Trailing data should be ignored.

All frame data must be stored consecutive without additional data. Thus the `i`th value specifies the byte offset to the beginning of the data for frame `i` and the `i+1`th value specifies the byte offset to the end of the data for frame `i` (as well as the beginning of the data for frame `i+1`).

Frame Data

The first four bytes are interpreted as `float` value, representing the time stamp. (This field is in present in format version 1.2 and later only.)

The second four byte of the frame data specify the number of particle lists as `unsigned int 32 bit` value.

Then the data for each list is written, beginning with a list header:

Bytes	Data	Description
0	byte	Vertex data type
1	byte	Color data type
2..5	float (32 bit)	Global Radius
[2|6]+0..3	4x byte	Global RGB Color
[2|6]+0..7	2x float (32 bit)	Global intensity color range
[2|6|+0..7	unsigned int 64 bit	Particle Count

The vertex data type is one of the following values:

Value	Vertex Data Type	Bytes per Particle	Description
0	`NONE`	`0`	No vertex data is stored
1	`FLOAT_XYZ`	`3*4=12`	For each vertex, the position is stored as three float co-ordinates
2	`FLOAT_XYZR`	`4*4=16`	For each vertex, the position is stored as three float co-ordinates and the sphere radius is stored as fourth float value
3	`SHORT_XYZ`	`3*2=6`	For each vertex, the position is stored as unsigned int 16 bit value, quantized to the rande [0.. 65535].

The color data type is one of the following values:

Value	Color Data Type	Bytes per Particle	Description
0	`NONE`	`0`	No color data is stored
1	`UINT8_RGB`	`3`	For each vertex, the color is stored as three unsigned bytes RGB color components
2	`UINT8_RGBA`	`4`	For each vertex, the color is stored as four unsigned bytes RGBA color components
3	`FLOAT_I`	`4`	For each vertex, the color is stored as one float intensity value.
4	`FLOAT_RGB`	`3*4=12`	For each vertex, the color is stored as three float RGB color components
5	`FLOAT_RGBA`	`4*4=16`	For each vertex, the color is stored as four float RGBA color components

If the vertex data type is `FLOAT_XYZ` or `SHORT_XYZ`, the global radius is stored as float. Otherwise this value is omitted. This value is used as radius for all particles.

If the color data type is `NONE` the global color is specified as four unsigned byte values RGBA color components to be used for all particles in this list (cf. color data type `UINT8_RGBA`).

If the color data type is `FLOAT_I` two float values are stored, specifying the minimum and maximum intensity values used in this particle list. The float color values of all particles must be inside this interval.

For all other color data type these values are omitted.

The particle count is specified as `unsigned int 64 bit`. It is safe to use the value 0 here, regardless of the specified vertex data type and color data type.

If the vertex data type is `NONE`, then the particle count must be zero and no particle data follows, regardless of the specified color data type.

Otherwise particle data is now written to the file as one block of interleaved data. All components of the particles are interleaved, position and color data (cf. array of structures). For example, for vertex data type `FLOAT_XYZ` and color data type `UINT8_RGB`, 15 bytes are written for each particle:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	…
Data	X1				Y1				Z1				R1	G1	B1	X2				Y2				…

The length of this particle data is specified by the values given above. However, a reader must not read beyond the frame data end specified in the seek table. If the frame data is not long enough the data must be considered corrupted. Any trailing data within the frame data block must be ignored.

MMPLD – Version 1.1 Extensions

This section only describes the extensions of MMPLD version 1.1 to version 1.0.

The file format version 1.1 is **not** an improvement over version 1.0 but a very application specific extension. **Usually, the file format version 1.0 is used.**

In the header data, the version number must be `101`.

After the particle data, additional cluster data is written for each particle list in each particle frame.

First, one `unsigned int 32 bit` value specifies the number of clusters stored. This can be zero.

A second value of type `size_t`[2] specifies the overall cluster data size in bytes for this particle list and time frame.

Then the specified number of bytes are stored, to be interpreted as implementation dependent cluster information.

[2] http://www.cplusplus.com/reference/cstddef/size_t/

Appendix A – Example Data Conversion Project

The following XML project file converts a sequence of multiple *IMD atom data files*[3] into a single MMPLD file. The required values for the modules need to be specified by a parameter file or corresponding command line arguments (using the option -v).

```xml
<?xml version="1.0" encoding="utf-8"?>
<MegaMol type="project" version="1.0">
  <job name="converterjobpldseries" jobmod="job">
    <module class="DataWriterJob" name="job" />
    <module class="MMPLDWriter" name="writer" />
    <call class="DataWriterCtrlCall" from="job::writer" to="writer::control" />
    <module class="DataFileSequence" name="seq">
      <param name="fileNameSlotName" value="data::filename" />
    </module>
    <module class="IMDAtomData" name="data">
      <param name="radius" value="2"/>
    </module>
    <call class="MultiParticleDataCall" from="writer::data" to="seq::outdata" />
    <call class="MultiParticleDataCall" from="seq::indata" to="data::getdata" />
  </job>
</MegaMol>
```

Working with other data sources requires simply to replace the module of type IMDAtomData and optionally removing the module DataFileSequence and rerouting the MultiParticleDataCall calls.

[3] http://imd.itap.physik.uni-stuttgart.de/userguide/header.html#atomheader