Documentation du logiciel de post-processing de Liggghts
Avancement du dévelopement

F. G.

3 juin 2014

Table des matières

Version majeure	Version mineure	Date	Modifications
1	0	28/03/2011	Création de la compression des L dump (dump atomique de base)
2	0	12/09/2011	Ajout de la compression des F dump (dump de force sur objet)
2	1	16/06/2012	Correction d'un bug lors de l'écriture des formats dénombrables (2 et +)
2	5	05/2014	Modifications internes assez importantes : Le coarse-graining passe en classe dump, les identifiants ont leur classe à part plus modulable (IDS), la classe Writing permet un écriture plus modulable.

TABLE 1 – Version importantes de PostProcessing

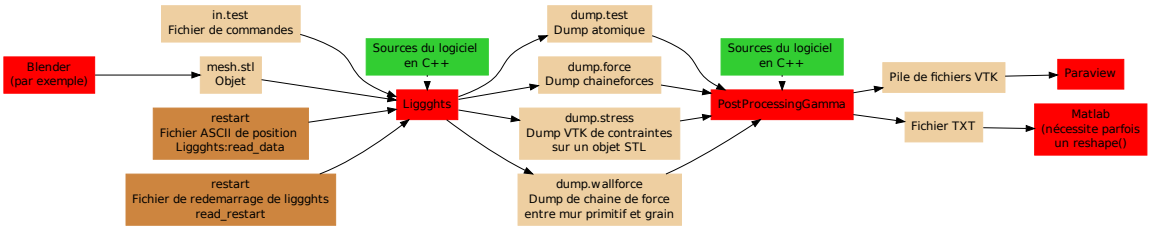

FIGURE 1 – Schéma de la chaîne de fonctionnement de Liggghts.

1 Historique

28/03/2011 : création de la compression de base pour les dump atomiques (positions des atomes, rayons etc. La compression se fait sur les valeurs à chaque timestep. Si au cours du temps (sur le fichier complet), le nombre de valeur pris par un type de donnée est fini (1, 2, 4 ou 16), les données sont compressées en allouant un nombre de bit inférieur à l'octet pour la donnée en question.
12/09/2011 : addition de la compression des dump de force sur les triangle de mesh. La compression se base le regroupement effectué auparavant dans le programme de post-processing des données de force et des données de dump atomique. Pour une backward compatibility correcte, les modifications à cette date sont en rouge dans ce fichier. Une extension apportée à la compression est la vérification s'il existe des modifications des données d'un timestep à l'autre, et l'ajout de type de données en lien avec cela.

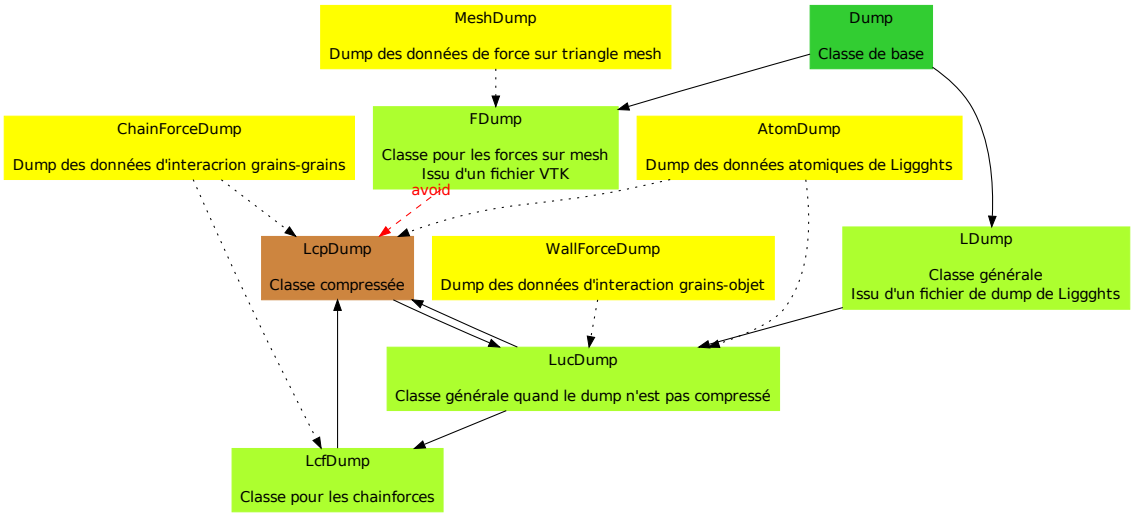

FIGURE 2 – Schéma des relations de classes de dumps dans PostProcessing OUTDATED

2 Fonctionnement général

Le dernier argument de la ligne de commande est le nom du fichier de données à lire. Il s'agît d'un dump de données atomique (données pour chaque atome), noté ATOMDUMP dans la suite, et qui correspond à une variable de classe LDump dans le code C++ (plus précisément LucDump si c'est un dump non compressé par le logiciel de PostProcessing, LcpDump si c'est un dump compressé). Voir la figure 2 pour les relations entre les classes du code source.

Lorsque l'argument `--vtk` est donné, le dernier argument doit être un fichier VTK unique contenant les données d'interactions entre un objet stl (mesh) et les grains (ie. les forces sur chaque triangle). Il s'agît alors d'un dump de classe C++ FDump.

Lorsque l'argument `--chainforce` est donné dans la ligne de commande, les 2 derniers arguments deviennent les fichiers de données. L'avant-dernier argument est un fichier ATOMDUMP, le dernier argument contient les chaines de forces, noté CFDUMP dans la suite, et correspond à une variable de classe LcfDump dans le code C++.

Dans certains cas (`--wallchainforce`, cf partie suivante), les 3 derniers arguments peuvent être des dumps, pour prendre en compte les chaines de forces primitivewall–grain, qui ne sont pas incluses dans les CFDUMP.

Pour la lecture du fichier, généralement, une première passe est effectuée pour repérer la position de départ des données de chaque pas de temps. Ces données sont stockées dans un fichier *.tmp qui permet une lecture plus rapide la fois suivante lorsque le même fichier de dump est réutilisé. Par contre, il faut régénérer le tmp si le dump est modifié ou lors de l'utilisation de CFDUMP (c'est fait automatiquement normalement).

A priori il est préférable de toujours utiliser les arguments `--v2` et `--rebuild` à partir de maintenant lorsque l'on utilise des chaines de force.

3 Exemple de quelques commandes ...

3.1 Commandes générales

`$ PostProcessingGamma --help null` : Liste tous les arguments possibles pour Post-Processing.

`$ PostProcessingGamma --dump2vtk --w/id --w/rayon ATOMDUMP` : transforme un dump en une série de fichiers VTK lisibles par paraview avec un champ contenant les ID et un champ contenant les rayons des atomes.

`$ PostProcessingGamma --chainforce --v2 --rebuild ATOMDUMP CFDUMP` : pour sortir des VTK affichant les chaines de forces dans paraview.

3.2 Pour un coarse graining correct

Pour le coarse-graining, il peut y avoir jusqu'à 3 dump à fournir :

- Coarse-graining de vitesses et compacités : un seul dump de type Ldump avec les positions des atomes (appelé ATOMDUMP dans la suite).
- Coarse-graining de contrainte : 2 dump avec l'argument `--chainforce`. Le dernier argument est un dump de type TCF (CFdump, CFDUMP dans la suite) et contient les forces particules-particules, l'avant dernier est un argument de type TL (Ldump) et contient les positions des atomes. Le coarse graining de vitesse est automatiquement effectué en même temps, mais il ne faut pas trop s'y fier je pense ...
- Coarse-graining de grains cylindriques polydisperses (2D) : utiliser l'argument `--polydisperse2D`.
- Coarse-graining incluant les interactions avec le mur : 3 dump avec les arguments `--chainforce --wallchainforce x x x`. Le dernier et l'avant dernier argument sont les dumps indiqué au point précédent, l'avant-avant dernier argument est un dump de type TL (LDump, WALLFORCEDUMP dans la suite) contenant les interactions grains-cylindre (ne fonctionne qu'avec un cylindre pour l'instant).

Problèmes à éviter :

- Faire un clean de tous les tmp pour les dump utilisés (par mesure de précaution, les tmp n'enregistrant pas les types de dump ça peut causer problème).
- Les timestep de tous les dump devraient être identiques pour éviter les problèmes (même si une petite vérification est faite pour les
- Il est conseillé d'utiliser toujours `--winsphere` en granulaire avec le ATOMDUMP, et de ne pas l'utiliser sinon (permet une mesure précise de la compacité). ATTENTION : il ne faut donc pas se fier aux compacités dans les coarse-graining de contrainte.
- Les contraintes sont TOUTES avec un signe MOINS (les données extraites sont donc $\sigma_{ij} = P\delta_{ij} - \tau_{ij}$.

Exemple de commandes :

```
$ PostProcessingGamma --coarse-graining 120 3 120 --use-box -0.1 0.1 -0.007 0.007 0 0
--winsphere --v2 ATOMDUMP

$ PostProcessingGamma --chainforce --coarse-graining 120 3 120 \
--use-box -0.1 0.1 -0.007 0.007 0 0.2 --periodicity ATOMDUMP CFDUMP

$ PostProcessingGamma --chainforce --coarse-graining 120 3 120 \
```

```
--use-box -0.1 0.1 -0.007 0.007 0 0.2 --v2  \
--wallchainforce 0 0.08 0.0025 WALLFORCEDUMP ATOMDUMP CFDUMP
```

Rappel des arguments des options :

```
--coarse-graining 120 3 120 : nb de boites en x, y, z
--chainforce : pas d'arguments
--use-box -0.1 0.1 -0.7 0.7 0 0.2 : limites de l'espace en -x, +x, -y, +y, -z, +z
--periodicity : le CFDUMP indique si les interactions traversent
                une condition limite periodique. (cf le tableau 1)
--v2 : indique un liggghts version 2 (qques diff\'erences de syntaxe ...).
                Implique periodicity
--wallchainforce 0 0.08 0.0025 : position x, position z et rayon du cylindre
                (rayon unused pour l'instant ...).
```

4 Extraction des données : la classe writing.

La classe Writing gère les fonctions particulières d'écriture des données dans les différents formats. À chaque démarrage de PostProcessing, elle est initialisée avec les informations suivantes :

- Le ou les formats de sortie :
  - vtk : format pour lecture paraview.
  - mat : format lisible avec matlab.
  - ascii : format tableau ascii.
- Pour chaque format la liste des variables à écrires (existantes a priori ou non).

4.1 Fonctionnement

Le principe est que lorsqu'une classe dump a besoin d'une écriture, elle converse avec la classe Writing à travers un échange de signaux (Writing est en fait un thread séparé). Donc Writing demande les infos donc il a besoin, et dump les charge en mémoire (si possible), et passe le pointeur à Writing. Sur les format VTK, qui ont 1 fichier par timestep, 1 seul est chargé à chaque fois. Par contre pour l'ascii et le mat, tout est chargé d'un coup ce qui peut être problématique pour des gros fichiers.

4.2 Compilation, exécution

Pour compiler avec le support pour l'écriture de fichiers matlab, il faut définir la macro MATLAB (idéalement dans le makefile), inclure les répertoires d'include de matlab dans la compilation, dans le linkage inclure les répertoires des librairies (a priori dynamiques) de matlab. Ensuite, lors de l'exécution, il faut que la librairie dynamique matlab soit dans les répertoires de recherche des librairies dynamiques (sous mac, il faut éditer la variable environnement $DYLD_LIBRARY_PATH).

4.3 Parler avec la classe Writing

En ligne de commande, l'argument après `--write` (ou `--W` qui est un raccourci) doit être une chaîne de caractères. Elle peut contenir des espaces, des retours à la ligne, des fabulations etc. qui seront ignorées. La chaîne est ensuite processée de la manière suivante :

- Les textes entre accolades { } sont des fonctions. Le principe est que le texte entre accolades est un nom de fonction, qui est appelé pour chaque variable suivante, en passant en outre le format d'écriture rencontré précédemment s'il existe. Voir le tableau 2 pour les fonctions existantes.
- Les textes entre crochets [] se réfèrent aux formats d'écriture. (e.g. `[+vtk]`, `[-mat]`, `[ascii]`). Un format précédé de + est séléctionné, de − est déséléctionné, de rien est conservé dans le même état (attention du coup).
- Le reste sont des variables, qui se réfèrent à la fonction ou à la classe précédente. Les variables sont séparées par des virgules ou des points virgules. Il y a deux variables particulières :
  - `def` : charger les variables par défaut pour le format en cours. Utile pour écrire un format non chargé automatiquement.
  - `nodef` : ne pas charger les variables par défaut. Utile pour modifier un format chargé automatiquement.

Ce qui donne `"cfdump:mat [cfdump:mat] vx, pos, masse, atm:masse"`.

Fonction	Effet	Exemple
`{expand}`	Les variables en interne s'expandent en variables de dimension 1	`[vtk]{expand}pos` est comme `[vtk]posx,posy,posz`

TABLE 2 – Fonctions définies pour l'instant

Variables de Writing n'existant pas en IDS	Dimension
`general1D` (Controle seulement)	1
`pos`	2
`v`	2
`f`	2
`omega`	2
`sigma`	3
`cff`	2
`cfforce`	2
`sigmak`	3
`sigmatot`	3
`forcewall`	2

TABLE 3 – Variables compatibles avec chaque classe. En Vert, extrait obligatoirement. En rouge, raccourci (vecteurs), en bleu, raccourci (tenseur)

4.4 Dimensions

Chaque variable a une dimension. Les variables de base telles que définies dans la classe IDS sont toutes de dimension 1. Writing défini des variables de dimension 2 (vecteur) et 3 (tenseur). Le principe est que les variables de dimensions $n > 1$ se décomposent en variables de dimension $n-1$ en ajoutant d'abord x, puis y, puis z.

- **vtk**
  - 1D SCALARS
  - 2D VECTORS
  - 3D TENSORS
- **mat** Pour les matrices. 1e dimension : atome / contact / boîte, 3e dimension : timestep, 2e dimension
  - 1D 1 sur la 2e dimension des matrices.
  - 2D 3 sur la 2e dimension des matrices.
  - 3D 9 sur la 2e dimension des matrices.
- **ascii** L'écriture ascii est un peu particulière. Si v le nombre de variables à écrire, N_i est le nombre d'atomes / contacts / boîtes pour la variable i, d_i la dimension de la variable i, ts le nombre de ts.
  - On ne peut avoir $ts > 1$ AND $\exists i \,|d_i > 1$ AND $N_i > 1$.
  - Si $ts = 1$ AND $\forall i, d_i = 1$ AND $N_i = cst$, alors 1 seul fichier est écrit (extension `-ascii.txt`) contenant toutes les variables.
  - Sinon, on écrit 1 fichier par variable (quelle que soit leur dimension).

NB : l'utilisation de la fonction `{expand}` rend toutes les variables de dimension 1 en interne, permettant l'écriture d'un seul fichier ascii tout en utilisant les noms raccourcis par exemple.

Type	Classe associée	ID min	ID max
TL	LDump	1	63
TCF	LcfDump	64	127
TF	FDump	128	191
TCOARSE	CoarseDump	192	254
TNONE		1	254

TABLE 4 – Les différents types de données définis par IDS et leur range d'identifiants. NB : Les ID sont définis en CHAR et sont donc limités á 256 valeurs

5 La classe IDS

La classe IDS définie les types de données pour tous les fichiers de dump. Elle est de plus extensible et peut ajouter des donnéese qu'elle ne connaît pas avec les propriétés de base. Chaque variable a un nom de base (en majuscule) et 1 ou plusieurs alias.

idname[192]="COARPHI" ; idname[193]="COARATM" ; idname[194]="COARCONTACTS" ; idname[195]="COARRAD" ; idname[196]="COARSIGKXX" ; idname[197]="COARSIGKXY" ; idname[198]="COARSIGKXZ" ; idname[199]="COARSIGKYX" ; idname[200]="COARSIGKYY" ; idname[201]="COARSIGKYZ" ; idname[202]="COARSIGKZX" ; idname[203]="COARSIGKZY" ; idname[204]="COARSIGKZZ" ; idname[205]="COARSIGTOTXX" ; idname[206]="COARSIGTOTXY idname[207]="COARSIGTOTXZ" ; idname[208]="COARSIGTOTYX" ; idname[209]="COARSIGTOT idname[210]="COARSIGTOTYZ" ; idname[211]="COARSIGTOTZX" ; idname[212]="COARSIGTOT idname[213]="COARSIGTOTZZ" ;

Constante c	**Int. c**	**Txt dump**	**Nom commun**	**Format**
ID	1	id	Identifiant atome	UInt
TYPE	2	type	Type d'atome	UChar
POSX	3	x	Position x	Float / Double
POSY	4	y	Position y	Float / Double
POSZ	5	z	Position z	Float / Double
VX	6	vx	Vitesse x	Float / Double
VY	7	vy	Vitesse y	Float / Double
VZ	8	vz	Vitesse z	Float / Double
RAYON	9	radius	Rayon	Float / Double
FX	10	fx	Force x	Float / Double
FY	11	fy	Force y	Float / Double
FZ	12	fz	Force z	Float / Double
MASSE	13	mass	Masse	Float / Double [b]
OMEGAX	14	omegax	Vitesse angulaire x [c]	Float / Double [b]
OMEGAY	15	omegay	Vitesse angulaire y [c]	Float / Double [b]
OMEGAZ	16	omegaz	Vitesse angulaire z [c]	Float / Double [b]
SIGMA[XYZ][XYZ]	17 à 25	sigma[xyz][xyz]	Contraintes [c]	Float / Double [b]
FORCEWALLX	26	f_force_cyl[1]	Force particule/mur	Float / Double [b]
FORCEWALLY	27	f_force_cyl[2]	Force particule/mur	Float / Double [b]
FORCEWALLZ	28	f_force_cyl[3]	Force particule/mur	Float / Double [b]
ATMPRESSURE	29	atmpressure	Pression par atome	Float / Double [b]
UNKNOWN	255	RESERVED	RESERVED	

a. Ne doit pas être compressé.
b. Compression non réellement testée.
c. Unused. Leur utilisation devrait être vérifiée si besoin.

TABLE 5 – Table des types de données de type TL définis au 3 juin 2014.

Constante c	Int. c	Txt dump	Nom commun	Format
CFID1	64	c_cout[1]	ID atm1	UInt
CFID2	65	c_cout[2]	ID atm2	UInt
CFFORCEX	66	c_cout[3] / c_cout[4] [d]	Force x du lien	Float / Double
CFFORCEY	67	c_cout[4] / c_cout[5] [d]	Force y du lien	Float / Double
CFFORCEZ	68	c_cout[5] / c_cout[6] [d]	Force z du lien	Float / Double
CFMAG	69	Construit	Magnitude du lien	
CFR	70	Construit	Lien en polaire [a]	
CFTHETA	71	Construit	Lien en polaire [a]	
CFPHI	72	Construit	Lien en polaire [a]	
CFX	73	Construit	Position x du lien [a]	
CFY	74	Construit	Position y du lien [a]	
CFZ	75	Construit	Position z du lien [a]	
CFPERIOD	76	c_cout[3] [d]	Périodicité du lien	UChar
CFID1X	77	c_contacts[1] [e]	Pos x atm 1 du contact	Float / Double
CFID1Y	78	c_contacts[2] [e]	Pos y atm 1 du contact	Float / Double
CFID1Z	79	c_contacts[3] [e]	Pos z atm 1 du contact	Float / Double
CFID2X	80	c_contacts[4] [e]	Pos x atm 2 du contact	Float / Double
CFID2Y	81	c_contacts[5] [e]	Pos y atm 2 du contact	Float / Double
CFID2Z	82	c_contacts[6] [e]	Pos z atm 2 du contact	Float / Double
UNKNOWN	255	RESERVED	RESERVED	

a. Ne doit pas être compressé.
b. Compression non réellement testée.
c. Unused. Leur utilisation devrait être vérifiée si besoin.
d. Dans les version récentes de liggghts, la périodicité est indiqué dans la coordonnée 3 (CFPERIOD). Les autres c_cout sont donc décallés de 1.
e. Peut entrer en conflit avec CFFORCE et CFID. Il faut faire attention.

TABLE 6 – Table des types de données de type TCF définis au 3 juin 2014.

Constante c	Int. c	Txt dump	Nom commun	Format
PRESSURE	128	pressure	Pression	float
SHEARSTRESS	129	shearstress	Cisaillement	float
FORCEX	130	forcesTri [a]	Force X	float
FORCEY	131	forcesTri [a]	Force Y	float
FORCEZ	132	forcesTri [a]	Force Y	float
NORMALEX	133	normales	Normale X	float
NORMALEY	134	normales	Normale Y	float
NORMALEZ	135	normales	Normale Z	float
POINTX	136	POINTS	Point X	float
POINTY	137	POINTS	Point Y	float
POINTZ	138	POINTS	Point Z	float
POLY1	139	POLYGONS	Nb de sommet du polygone	Int
POLY2	140	POLYGONS	Sommet 1	Int
POLY3	141	POLYGONS	Sommet 2	Int
POLY4	142	POLYGONS	Sommet 3	Int
STRESSX	143	stress [b]	Force X	float
STRESSY	144	stress [b]	Force Y	float
STRESSZ	145	stress [b]	Force Y	float
CENTREX	146	CELL [b]	Normale X	float
CENTREY	147	CELL [b]	Normale Y	float
CENTREZ	148	CELL [b]	Normale Z	float
UNKNOWN	255	RESERVED	RESERVED	

a. Extraites d'une modification des sources de Liggghts 1. Plus utilisés.

b. Pour la version 2 de liggghts. La commande `-rebuild` permet de reconstruire FORCE[XYZ] et d'autres choses à partir des données de v2.

TABLE 7 – Table des types de données de type TF définis au 3 juin 2014 pour les FDump (dump de force sur les triangles d'un mesh, lus dans un fichier VTK).

Constante c	Int. c	Txt dump	Nom commun	Format
COARPHI	192	Calculé	Compacité	float
COARATM	193	Calculé	Nb d'atomes	float
COARCONTACTS	194	Calculé	nb de contacts	float
COARRAD	195	Calculé	Rayon	float
COARSIGKXX	196	Calculé	Contrainte cinétique xx	float
COARSIGKXX	197	Calculé	Contrainte cinétique xy	float
COARSIGKXX	198	Calculé	Contrainte cinétique xz	float
COARSIGKXX	199	Calculé	Contrainte cinétique yx	float
COARSIGKXX	200	Calculé	Contrainte cinétique yy	float
COARSIGKXX	201	Calculé	Contrainte cinétique yz	float
COARSIGKXX	202	Calculé	Contrainte cinétique zx	float
COARSIGKXX	203	Calculé	Contrainte cinétique zy	float
COARSIGKXX	204	Calculé	Contrainte cinétique zz	float
COARSIGTOTXX	205	Calculé	Contrainte totale xx	float
COARSIGTOTXX	206	Calculé	Contrainte totale xy	float
COARSIGTOTXX	207	Calculé	Contrainte totale xz	float
COARSIGTOTXX	208	Calculé	Contrainte totale yx	float
COARSIGTOTXX	209	Calculé	Contrainte totale yy	float
COARSIGTOTXX	210	Calculé	Contrainte totale yz	float
COARSIGTOTXX	211	Calculé	Contrainte totale zx	float
COARSIGTOTXX	212	Calculé	Contrainte totale zy	float
COARSIGTOTXX	213	Calculé	Contrainte totale zz	float
UNKNOWN	255	RESERVED	RESERVED	

a. Extraites d'une modification des sources de Liggghts 1. Plus utilisés.

b. Pour la version 2 de liggghts. La commande `-rebuild` permet de reconstruire FORCE[XYZ] et d'autres choses à partir des données de v2.

TABLE 8 – Table des types de données de type TCF définis au 3 juin 2014.

6 Description du format de fichier de compression de dump

Le format dump compressé permet de diminuer la taille utilisée par les dump, en gardant comme format de sortie de Liggghts des fichiers ASCII pour lesquels les traitements de base de fichiers caractères fonctionnent. Les différences entre le fichier après décompression et le fichier original doivent être nulles autant que faire ce peut (*compression lossless*).

6.1 Header

- String : "AVFFLIGGGHTSDUMPCOMPRESSED" (26 octets). String "AVFFLIGGGHTSDUMPCOMPRES2.0" (26 octet).
- Unsigned Short Integer : nombre de caractère dans le nom de fichier original (2 octets)
- String : nom du fichier original
- Integer : nombre de timestep (4 octets)
- Unsigned Char : nombre de données dumped par par de temps (1 octet). **Les données écrites doivent être les mêmes pour tous les pas de temps**
- Pour chaque donnée :
  - Unsigned char : Type de donnée (id, radius, x ...) (1 octet). Table 6.
  - Unsigned char : Format de donnée (float, double etc.). (1 octet)
  - Pour les formats dénombrables, s'ensuit la liste complète des valeurs. Toute la liste doit être renseignée, y compris les valeurs non utilisées (par exemple si seulement 3 valeurs sur 4 sont utiles pour un type FLOAT_DENOM_4, la 4e doit tout de même être renseignée avec un float quelconque).
  - Si le format est un des format du tableau 9 avec le flag MASK_ALWAYS_THE_SAME=128 (*eg.* FLOAT | MASK_ALWAYS_THE_SAME) : tous les timestep ont les mêms valeur pour ce champ. Ce champ ne sera donc renseigné qu'au premier timestep, et il ne faudra pas le lire ensuite.

6.2 Corps du fichier

Le corps du fichier est plus simple que l'header.

- Unsigned Integer : valeur du timestep (4 octets) pour un TL. Pour un TF, nombre de points.
- Unsigned Integer : nombre de particules (4 octets) pour un TL. Pour un TF, nombre de triangles.
- Float ×6 : les bords de boîtes. Dans le cas d'un dump de type force (ie. un vtk), ces champs sont mis à 0.
- Données complètes dans l'ordre donné dans l'header. Les types définis constants dans l'header sont ignorés. Les types ayant le MASK_ALWAYS_THE_SAME ne sont donnés qu'au premier timestep. Pour les types de taille inférieure à l'octet, le dernier octet entamé est completé par des bits de faible poids ignorés à la lecture, qui n'empiètent pas sur les données suivantes.

Constante c	Numéro de définition c	Nom commun	Taille (bit/octet)
CHAR	1	char	8 / 1
UCHAR	2	unsigned char	8 / 1
SINT	3	short int	16 / 2
USINT	4	unsigned short int	16 / 2
INT	5	int	32 / 4
UINT	6	unsigned int	32 / 4
LINT	7	long int	64 / 8
ULINT	8	unsigned long int	64 / 8
FLOAT	9	float	32 / 4
DOUBLE	10	double	64 / 8
CHAR_CST	17	char constant	0
UCHAR_CST	18	unsigned char constant	0
SINT_CST	19	short int constant	0
USINT_CST	20	unsigned short int constant	0
INT_CST	21	int constant	0
UINT_CST	22	unsigned int constant	0
LINT_CST	23	long int constant	0
ULINT_CST	24	unsigned long int constant	0
FLOAT_CST	25	float constant	0
DOUBLE_CST	26	double constant	0
CHAR_DENOM_2	33	char constant	1 / 0
UCHAR_DENOM_2	34	unsigned char constant	1 / 0
SINT_DENOM_2	35	short int constant	1 / 0
USINT_DENOM_2	36	unsigned short int constant	1 / 0
INT_DENOM_2	37	int constant	1 / 0
UINT_DENOM_2	38	unsigned int constant	1 / 0
LINT_DENOM_2	39	long int constant	1 / 0
ULINT_DENOM_2	40	unsigned long int constant	1 / 0
FLOAT_DENOM_2	41	float constant	1 / 0
DOUBLE_DENOM_2	42	double constant	1 / 0
CHAR_DENOM_4	49	char constant	2 / 0
UCHAR_DENOM_4	50	unsigned char constant	2 / 0
SINT_DENOM_4	51	short int constant	2 / 0
USINT_DENOM_4	52	unsigned short int constant	2 / 0
INT_DENOM_4	53	int constant	2 / 0
UINT_DENOM_4	54	unsigned int constant	2 / 0
LINT_DENOM_4	55	long int constant	2 / 0
ULINT_DENOM_4	56	unsigned long int constant	2 / 0
FLOAT_DENOM_4	57	float constant	2 / 0
DOUBLE_DENOM_4	58	double constant	2 / 0
CHAR_DENOM_16	65	char constant	4 / 0
UCHAR_DENOM_16	66	unsigned char constant	4 / 0
SINT_DENOM_16	67	short int constant	4 / 0
USINT_DENOM_16	68	unsigned short int constant	4 / 0
INT_DENOM_16	69	int constant	4 / 0
UINT_DENOM_16	70	unsigned int constant	4 / 0
LINT_DENOM_16	71	long int constant	4 / 0
ULINT_DENOM_16	72	unsigned long int constant	4 / 0
FLOAT_DENOM_16	73	float constant	4 / 0
DOUBLE_DENOM_16	74	double constant	4 / 0

Table 9 – Table des formats de données définis au 3 juin 2014.