CGNS Standard Extension to Support Particle Data

CPEX0046 v2.0 revision 1

Alexandre Minot, Arun Rao - CONVERGE CFD

alexandre.minot@convergecfd.com, arun.rao@convergecfd.com

Revisions

1. Added email addresses

Introduction

The CGNS standard does not offer a practical way to hold particle data. This means that CFD codes typically use a different file format when exporting results from simulation of particle-laden flows. An initial attempt was made by Thomas Hauser to include particle data in the CGNS standard, but the proposal was not completed.

This document supports a new proposal for inclusion of particle data in CGNS. This new proposal includes modification to the SIDS, MLL, FMM and an example case. We are also working with Tecplot and Kitware to get the Tec360 and VTK/ParaView CGNS readers updated. We believe this new proposal covers the needs of the original proposal, while offering a complete set of modifications to the CGNS standard.

To give a bit more context, this proposal is already implemented in the development version of our code, CONVERGE CFD, in order to validate the format. We are able to use it to export results for all of our simulations. For example, this new data structure supports exporting simulation results of a piston engine simulation, where fuel is injected in the form of liquid particles, evaporates, and combusts using detailed chemistry mechanisms.

Scope of this document

Most of the information describing the extension is contained in the modified SIDS and file mapping. The present document is intended to provide a summary of all the modifications, along with explanations of why we made certain choices. We will also attempt to answer questions that were raised when the CPEX0046 v1.0 was submitted.

If you find discrepancies in node definitions between this document and the SIDS, please trust the SIDS. As the community and steering committee offer ideas on the best way to implement the particle extension, we will be updating the SIDS, file mapping and MLL. This document might therefore become out of date.

Description of new nodes

A new node is added to the *Base_t* to hold particle data. This node, of type *ParticleZone_t,* contains most of the modifications needed in this extension.

ParticleZone_t node

The *ParticleZone_t* node contains data pertaining to a given set of particles. It is a child of the *Base_t*. There can be multiple *ParticleZone_t* nodes in a base. The intent is to offer the ability to create different groups of particles, for instance, solid particles and liquid particles, or blue particles and red particles. *ParticleZone_t* can have *FamilyName_t* and *AdditionalFamilyName_t* to help with their differentiation.

Much like *Zone_t*, *ParticleZone_t* contains *PatricleCoordinates_t* and *ParticleSolutions_t*. There can be multiple of those and they are linked to the simulation time using *ParticleIterativeData_t*.

```
ParticleZone_t< ParticleSize > :=
   {
   List( Descriptor_t Descriptor1 ... DescriptorN ) ;                (o)

   List( ParticleCoordinates_t<ParticleSize>
         ParticleCoordinates MovedParticles1 ... MovedParticlesN ) ;  (o)

   FamilyName_t FamilyName ;                                         (o)

   List( AdditionalFamilyName_t AddFamilyName1 ... AddFamilyNameN ) ; (o)

   List( ParticleSolution_t< ParticleSize>
         ParticleSolution1 ... ParticleSolutionN ) ;                 (o)

   List( IntegralData_t IntegralData1 ... IntegralDataN ) ;          (o)

   ParticleIterativeData_t<NumberOfSteps> ParticleIterativeData ;     (o)

   ReferenceState_t ReferenceState ;                                 (o)

   DataClass_t DataClass ;                                           (o)

   DimensionalUnits_t DimensionalUnits ;                             (o)

   ParticleEquationSet_t ParticleEquationSet ;                       (o)

   List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ;  (o)
   } ;
```

If the *ParticleZone_t* is so similar to a *Zone_t*, why create a new node? Zones are dedicated to holding a simulation mesh. As in, elements connected to each other. Each particle is independent of other particles. It has no neighbors. Creating a dedicated node for particles makes the data structure clearer, and helps users identify data quickly.

Because *Zone_t* is dedicated to holding meshes, we also decided to put the *ParticleZone_t* at the *Base_t* level, as opposed to inside the *Zone_t*. Both options have advantages and drawbacks. Putting the *ParticleZone_t* inside the *Zone_t* has the advantage of designating that the particles belong to a *Zone_t*. In particular, they would inherit the spatial bounding box of the *Zone_t*, which can be helpful for the reader. But, is that really an advantage though? What if the *Zone_t* distribution corresponds to the MPI distribution on the HPC? Does it still make sense to force the particles to follow that distribution? In all likelihood, particles will also be subject to the same MPI distribution, so it will probably still be easy to write them inside zones, but does it make sense for the reader?

Another way to see that it is better for particles to be outside the *Zone_t* is to consider the case where we have particles only and no mesh. How then do we create a *Zone_t*? What do we put for *ZoneType* and *IndexDimension*? *Unstructured*, [0, 0]? We could, but the reader would have to deliberately ignore that information, which goes against the principle of a data structure. It is for these reasons that we have decided to have the *ParticleZone_t* be a child of the *Base_t*.

The order of particles in a *ParticleZone_t* carries no meaning. If a particle is first at a given time, there is no guarantee that the first particle at another time is the same particle. Also, the number of particles can change over time. This means that particles can break up or aggregate, but the mechanism is not explicitly traced by the data structure. If particle 0 breaks up between instant 0 and instant 1, there is simply one more particle in that data structure at instant 1. If a blue particle aggregates with a red particle to form a purple particle, there is simply one less blue particle, one less red particle, and one more purple particle. There is also no global IDs of particles in the current proposal. We think that the necessity for global IDs is rare. Even though it is not ideal, if global IDs are necessary for a specific application, they can be added as a *DataArray_t* in the *ParticleSolutions_t* alongside results.

Though the order carries no meaning, particle solutions in the *ParticleSolutions_t* must follow the same order as the coordinates (unless a *PointList* is defined).

ParticleZone_t nodes and *Zone_t* nodes are independent. There is no implicit meaning that particles in a *ParticleZone_t* need to be carried by a flow defined in a *Zone_t*. Simulation results can be fully defined by a *Base_t* and a *ParticleZone_t* if the solver does not solve equations in an Eulerian framework (or if the user does not wish to export Eulerian data). In particular, we expect SPH codes to be able to use our extension.

ParticleCoordinates_t

The *ParticleCoordinates_t* node holds the location of the centers of particles. It is a child of *ParticleZone_t.* Particles are implicitly defined as 1D elements. They are not connected to each other, so no connectivity logic is needed. All that is needed is a list of coordinates.

```
ParticleCoordinates_t< int ParticleSize, int PhysicalDimension> :=
   {
   DataArray_t<DataType,PhysicalDimension, 2> BoundingBox ;           (o)

   List( Descriptor_t Descriptor1 ... DescriptorN ) ;                 (o)

   List( DataArray_t<DataType, ParticleSize>
         DataArray1 ... DataArrayN ) ;                                (o)

   DataClass_t DataClass ;                                            (o)

   DimensionalUnits_t DimensionalUnits ;                              (o)

   List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ;  (o)
   } ;
```

ParticleCoordinates_t and children *DataArray_t* inherits *ParticleSize_t,* the number of particles, from *ParticleZone_t*, so no *DataSize* function is required. Users are encouraged to use coordinate variable names already defined in appendix A and no extension to appendix A is needed. In particular, we discourage the use of *ParticleCoordinateX* and similar as coordinate variable names.

ParticleCoordinate_t allows the definition of a bounding box, which is why *PhysicalDimension* is inherited.

ParticleSolution_t

ParticleSolution_t is a child of *ParticleZone_t* and holds the solution on each particle.

```
ParticleSolution_t< int ParticleSize> :=
   {
   List( Descriptor_t Descriptor1 ... DescriptorN ) ;                (o)

   IndexRange PointRange ;                                           (o)
   IndexArray<DataSize[], int> PointList ;                           (o)

   List( DataArray_t<DataType, DataSize[]>
         DataArray1 ... DataArrayN ) ;                               (o)

   DataClass_t DataClass ;                                           (o)

   DimensionalUnits_t DimensionalUnits ;                             (o)

   List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ;  (o)
   } ;
```

A *PointRange* or *PointList* is allowed should the user want to define solutions for only a subset of particles. We therefore need a *DataSize* function, that returns *ParticleSize* if no *PointRange* or *PointList* is present, or the number of particles corresponding to *PointList* or *PointRange*. If the community feels like this function should be called *ListLenght* instead of *DataSize*, we can make the change.

We have added the variable names *Mass, Radius* and *Diameter* to appendix A as these are typical solutions of particle simulation.

ParticleEquationSet_t

In order to describe models linked to particles, we have created new model and equation nodes. These nodes are contained in *ParticleEquationSet_t* which can be a child of *Base_t* and *ParticleZone_t.*

```
ParticleEquationSet_t :=
   {
   List( Descriptor_t Descriptor1 ... DescriptorN ) ;                (o)

   int EquationDimension ;                                           (o)

   ParticleGoverningEquations_t; ParticleGoverningEquations ;        (o)

   ParticleCollisionModel_t ParticleCollisionModel ;                 (o)

   ParticleBreakupModel_t ParticleBreakupModel ;                     (o)

   ParticleForceModel_t ParticleForceModel ;                         (o)

   ParticleWallInteractionModel_t ParticleWallInteractionModel ;     (o)

   ParticlePhaseChangeModel_t ParticlePhaseChangeModel ;             (o)

   DataClass_t DataClass ;                                           (o)

   DimensionalUnits_t DimensionalUnits ;                             (o)

   List( UserDefinedData_t UserDefinedData1 ... UserDefinedDataN ) ; (o)
   } ;
```

ParticleGoverningEquations is a new enum containing *DEM*, *DSMC* and *SPH*. We created this node because *GoverningEquations* inherits *CellDimension*, which is not defined in the *ParticleZone*. If the community is not concerned with that, we can simply allow the original *GoverningEquations* nodes in *ParticleZone_t.*

We also created enums to describe breakup models, collision models, force models (lift and drag models), wall interaction models, and phase change models. The full list of added models is: *Linear, NonLinear, HardSphere, SoftSphere, LinearSpringDashpot, Pair, BaiGosman, HertzMindlin, HertzKuwabaraKono, Kuhnke, ORourke, Wruck, Stochastic, NonStochastic, NTC, KelvinHelmholtz, KelvinHelmholtzACT, RayleighTaylor, KelvinHelmholtzRayleighTaylor, ReitzKHRT, TAB, ETAB, LISA, SHF, PilchErdman, ReitzDiwakar, Sphere, NonShpere, Tracer, BeetstraVanDerHoefKuipers, Ergun, CliftGrace, Gidaspow, HaiderLevenspiel, PlessisMasliyah, SyamlalOBrien, SaffmanMei, TennetiGargSubramaniam, Tomiyama, Stokes, StokesCunningham, WenYu, Boil, Condense, Flash, Nucleate, Chiang, Frossling, FuchsKnudsen.*

Because the *ParticleWallInteractionModel_t* can only be at the base level or at the particle level (and not at the wall level), it cannot be used to set different models at different walls. To do so, we invite users to use the *WallFunction_t* node, which has been defined with a loose definition specifically to host many models. If the community does not find this sufficient, we could allow *ParticleWallInteractionModel_t* to be a child of *BCProperty_t.*

Similarly, the current proposal does not provide a way to set different collision models for interactions between particles stored in different *ParticleZones_t.* That is, we cannot specify a specific collision model between red and green particles and a different one between red and blue particles. We have found no simple solution for this issue.

Conclusion

We have spent much time trying to figure out the best way to extend the CGNS standard to hold particle data. We find that our proposal is a good intermediary between simplicity and clarity. Because the new nodes that we propose mimic existing nodes in the *Zone_t*, implementation of our extension is straightforward. For example, once we wrote the SIDS and MLL, it only took us a few hours to implement the CGNS particle export in our code alongside our CGNS Eulerian export.

On top of this document, we are providing modifications to the SIDS, MLL, and FMM through the code repository, as well as an example case. We believe that this constitutes a complete proposal. If it is not, please let us know so we might provide additional support.

If the community seems receptive to our proposal, we will move forward on getting the Tec360 and VTK/ParaView CGNS readers updated, so they might be ready to test as our proposal awaits acceptance.