General Geometry Description Design & Status

Brett Viren

Physics Department

BROOKHAVEN NATIONAL LABORATORY

LBNE S&C Meeting Oct 2014

Outline

LBNE Geometry Landscape

There are:

Many things to model:

- FD, ND, 35t, CERN FST, ν-Beamline

Many reasons to model:

- Simulation, Detector Design, Reconstruction, PR Figures

Many technologies are model consumers:

- Geant4, ROOT, GraXML, custom validation code

Etc:

- Stamping and tracking of model versions.
- Parallel development of models.
- Parameter provenance (why a given model has a given feature).

Related LBNE S&C Requirements

LBNE Requirements for Geometry:

- Develop a data management system from which applications may derive their geometry information.
- Formats and interfaces must not be restricted to a single software package or toolkit.
- Accommodate non-ideal, "distorted" or "perturbed" versions, eg. as-built.
- Must support revision control.

Best practices:

- Allow multiple developers to work independently on unrelated parts
- Allow for parallel, competing geometries
- Limit "degrees of freedom" for non-experts, allow them for experts.
  - $\rightarrow$ easy things should be easy, hard things should be possible

LBNE Plans for Geometry

- Plans development based on extensive experience in geometry for STAR, Daya Bay and others.
- Driven by the high level requirements.
- Centered on developing a **geometry authoring system** independent from existing geometry systems but with functionality to export to required formats.
- Plan was presented as part of the BNL contribution to LBNE S&C at the May review.

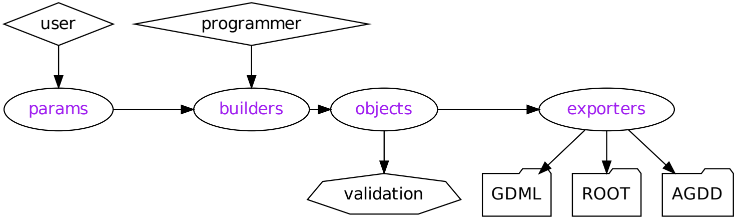
GeGeDe high level features

The **General Geometry Description** (GeGeDe) provides:

authoring	a human-oriented way to develop geometry descriptions
persistence	keeping geometry information for later use
provenance	what geometry was used, where it came from
visualization	methods to graphically represent the geometry
validation	checks for correctness
conversion	converting between different representations

Some of these aspects are still in development.

Major Design Components of GeGeDe



params	high-level, user-accessible **configuration** parameters
builders	reusable and hierarchical “chunks” of code responsible for **constructing** portions of the overall geometry
objects	comprehensive system for describing **Constructive Solid Geometry** objects (shapes, volumes, placements)
exporters	conversions from objects to various **exchange formats** including GDML, ROOT and AGDD.

Example Configuration

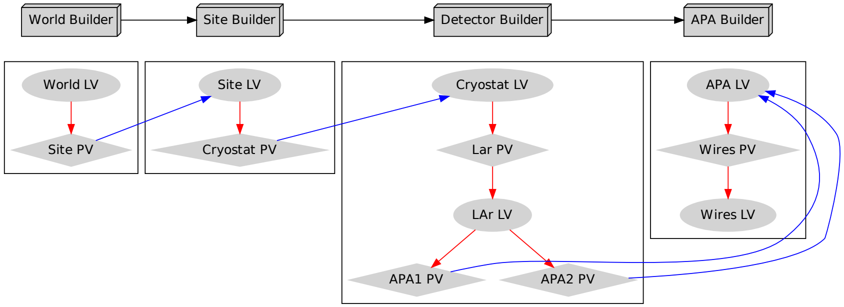
```
[everything]
class = mymodule.mybuilders.WorldBuilder
subbuilders = ["farsite", "nearsite", "testhall"]
size = Q("1000km")

[farsite]
class = mymodule.mybuilders.SiteBuilder
subbuilders = ["lardet"]
wireangle = Q("45*deg")
# etc, ...
```

- Each section names a **builder instance**.
- The `class` and `subbuilders` are the only reserved keys.
    - Interpreted by the `gegede.main` management layer.
    - Each builder free to expect/interpret any additional keys.
- Use `Q("...")` (`Q` for “quantity”) for expressing units.
    - Units strongly enforced!

Builders

- Reusable chunk of code that "knows" how to **construct** some geometry and may delegate some construction to sub-builders.
- Each builder:
  - exposes one or more logical volumes (LV) to the parent builder.
  - properly places (PV) any subbuilder's LVs into its own LVs
- Builders may explicitly create subbuilders or rely on implicit creation via the configuration layer.



Builder details

```
class MyBuilder(gegede.builder.Builder):
  defaults = dict(dx='1m', ...)

  def configure(self, dx='1m', dy='2m', dz='3m', **kwds):
    # ... specify any default/expected configuration
    # ... incorrect units are caught as errors!
    # ... configure self, typically trivially
    self.size = (dx,dy,dz)

  def construct(self, geom):
    # ... construct geometry with ``geom'' object
    box = geom.shapes.Box(self.name+'_shape', *self.size)
    # ... make LV with box, etc
    for sb in self.builders:
      # place top-level LV of any sub-builders
```

`defaults`	simple declaration of default parameters.
`configure()`	optional method for any post-processing of parameters.
`construct()`	actual construction using configuration. Any subbuilders will have already done their construction so builder may use their LVs

Geometry Objects

GeGeDe objects are patterned after Geant4 but are otherwise **independent**. They cover the usual categories:

shapes	(aka “solids”) such as box, tubs, sphere, etc
matter	elements, isotopes, mixtures, materials, etc
structure	rotations, positions, logical and physical volumes

Geometry objects follow an explicit **schema** that asserts:

- the above class categories
- class names and their expected data members
- prototypes for every data member
    - prototypes assert **units**, where appropriate
    - instances **must** use compatible units or exception raised!

Exporters

Exporters produce alternative representations of the objects:

GDML	for Geant4 in LArSoft/g4lbne. Initial exporter exists but not complete. Uses `lxml.etree` for assuring valid XML.
ROOT	for Geant4-free contexts such as data reconstruction. (to do, will bring in optional dependency on PyROOT)
AGDD	for feeding to the useful GraXML viewer (to do)
etc...	exporters are easy to develop. The initial GDML exporter took ∼2 hours to write.

- Consider exporter output as **transient data transfer** only, and not subject to downstream modification.
- The **builder code + configuration file** is definitive.

GGD: Current Functionality and To Do items

https://github.com/brettviren/gegede

Current functionality:

- Usable now, lots of tests, examples and other docs
- Provides valid GDML export (sometimes **too** valid)

ROOT	import and visualization, found several “warts”, **direct ROOT** export planned to overcome them.
Geant4	import and visualization, simple, **stand-alone visualization** program in development
GraXML	import and visualization, some GDML limitations (no assembly objects - **AGDD export** will help)

Todo:

- Flesh out collection of supported “shape” objects.
- Add support for optical surfaces and material properties.
- Add support for G4 sensitive detectors.
- Register package in PyPI for easier distribution.

35t modeling in GeGeDe

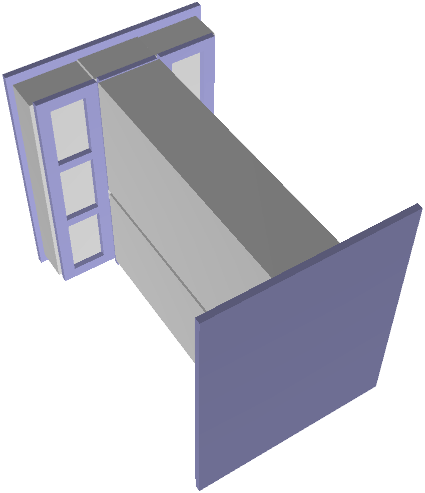
`https://github.com/LBNE/lbne-geometry`

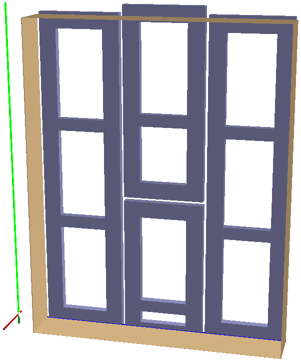
Initial goals for the `lbne-geometry` package:

- provide a vehicle to flesh out and polish GeGeDe itself.
- provide a working geometry for actual 35t usage with latest engineering dimensions.
- provide a single repository to hold all LBNE geometry information (although this may be too monolithic)
- provide full-featured example for other groups to follow.
- start work on remaining issues such as installation, versioning, provenance.

I plan to hand the 35t model over to geometry experts (Tyler) for longer term improvements and help with integration with LArSoft.

Some pictures of current model $\rightarrow$





Current 35t geometry in GeGeDe with large drift regions made invisible (left) and wire frames + field cage (right). Not shown is surrounding concrete/foam box. Rendered with ROOT/OpenGL.

35t GeGeDe work still to do

- Still getting dimensions right:
  - Utilizing Rahul's expertise as keeper of the model.
  - Just started mining PSL drawings on `http://webfs.psl.wisc.edu`
- Add PD scintillator paddles.
- Add wire planes with option to turn them on/off
  - LArsoft needs both versions. The "no wires" version is given to G4.
  - Very preliminary talks with Tyler about how to maybe do wire look ups in code w/no explicit wire volumes.
- Integration/installation
  - GeGeDe and `lbne-geometry` can right now be installed as simple, independent Python packages.
  - Will likely package the generated GDML as a small UPS product and have `lbnecode` depend on it.

Longer term to do items for `lbne-geometry`:

- Hand off 35t description to Tyler for refinement
- Reach out to Beam/ND/FD/FST groups