General Geometry Description
The gegede Package

Brett Viren

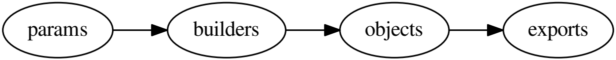
Physics Department

BROOKHAVEN NATIONAL LABORATORY

Outline

In a Nutshell

General **Ge**ometry **De**scription:[1]



- Simple **pipeline** of geometry information processing.
- Uses a simple system for **authoring** geometry descriptions.
- Authors write the description using a mix of:
    - A simple configuration language (**params**)
    - Structured Python code (**builders**)
- Produces in-memory **objects** adhering to a *Constructive Solid Geometry* (CSG) schema,
- Which are finally **exported** to variety of formats.

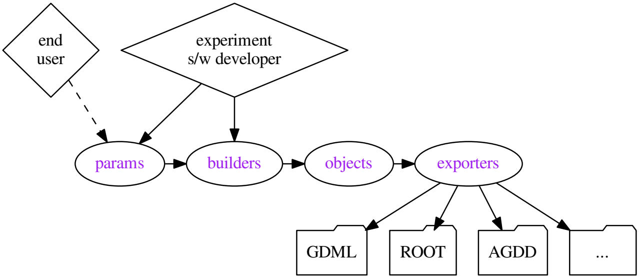
[1] GeGeDe or GGD. Pronounce it however you wish.

Non-features

GeGeDe is focused on authoring. Non-features include:

- × **No tracking** or other geometry querying algorithms.
  - × It is **not** a replacement for Geant4, nor ROOT's TGeo/GEOM
  - ✓ But, it can produce geometry data for them.
- × **No geometry content validation**, eg overlapping volumes.
  - ✓ But, other apps can check the exported geometry.
  - ✓ And, it will assure valid output formats.
  - ? Trivial hooks exist to add content validation in the future.
- × **No built-in visualization** services.
  - ✓ But, some of its formats can be visualized by other applications.
- × **No interactive/GUI** for model editing. It is not CAD.
  - ✓ But, has some experimental export format support for FreeCAD

Design Overview



params High-level, human-centric configuration language.
- Provided by experiment s/w developers, fiddled with by end users.

builders Structured, procedural geometry construction code.
- Experiment developers write this code.

objects In-memory representation of full geometry.
- Following strict schema defined in GeGeDe.

export Conversion to format suitable for some application.
- Batteries included with GeGeDe, or user-provided modules.

Configuration

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

[farsite]
class = mymodule.mybuilders.SiteBuilder
subbuilders = ["lardet"]
wireangle = Q("35*deg")
# etc, ...
```

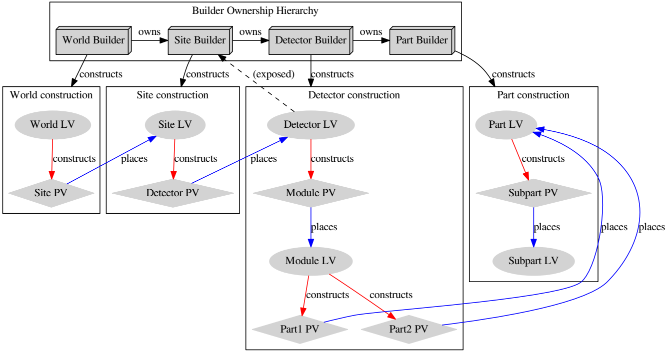
- Each section binds **name** and **parameter set** to a **builder instance**.
- One special builder provides the "world" logical volume.
  - File can configure multiple world builders.
  - First one listed "wins", or can specify one on the command line.
- The `class` and `subbuilders` are the only reserved keys.
  - The only two keywords reserved by GeGeDe internals.
  - Builders are free to require any additional key/value pairs (*eg*. `size, wireangle`).
- Use `Q("...")` for expressing units[2] (`Q` short for "quantity").

[2] http://pint.readthedocs.org/

Builders - GeGeDe's main code structure

- Builders written as **experiment-provided** code:
  - It does not "live" in the `gegede` Git repo!
  - Builders are subclasses of `gegede.builder.Builder` (or suitably duck-typed)
- Each builder:
  - Responsible for constructing some portion of the geometry.
  - Exposes zero or more logical volume (LV) objects to the **parent builder**.
  - Properly places the LVs of its **daughter builders** (*subbuilders*) into its own LVs.
- Builders typically are written as a **cooperative hierarchy**:
  - Builders can delegate to subbuilders.
  - Explicit subbuilder creation is allowed but better flexibility is achieved by listing then with the `subbuilders` configuration keyword.
  - Typically follow a loosely-coupled design but some collusion or "software contract" can be useful.
  - Arbitrary complex builder associations are allowed. A 1-parent/*n*-children tree is common.

Example Builder Hierarchy



- **Factor** each builder based on **geometry symmetries**.
- Builders directly construct “local” geometry.
- Hierarchy design best to follow loosely-coupled software contract:
  - Child builder **exposes** select LV(s) such that,
  - Parent builder knows how/where to **place** them.
- Allows for:
  - Decoupled development and reuse of builders.
  - Test “worlds” narrowed to a subbuilder’s exposed LVs

Builder Code Example

```
class MyBuilder(gegede.builder.Builder):

  def configure(self, dx='1m', dy='2m', dz='3m', **kwds):
    # Receive user configuration, providing defaults.
    # Incompatible units are caught as errors.
    # Here, configure self, in some way.
    self.size = (dx,dy,dz)

  def construct(self, geom):
    # Do local construction using ``geom'' object as only interface.
    box = geom.shapes.Box(self.name+'_shape', *self.size)
    # Add ``top-level'' LV to .volumes list for our parent to use.
    self.add_volume(box)
    # Place LVs from our child builders, guaranteed already constructed.
    for sb in self.builders:
      for sv in sb.volumes:
        # ... place ``sv'' in ``box'' ...
```

Geometry Objects

GeGeDe objects follow the CSG model:

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

These objects:

- Reference each other by name to form some graph.
  - Name-based references used to parallel GDML's schema.
  - May explicitly name objects or GeGeDe will generate a unique name.
- Are represented as Python `namedtuple` instances.
- Objects follow a specific schema

Schema Definition Language

```
Schema = dict(
  shapes = dict(
    Box = (("dx","1m"), ("dy","1m"), ("dz","1m")),
    # ...
  matter = dict(
    Element = (("symbol",str), ("z",int), ("a","0.0g/mole")),
    # ...
  structure = dict(
    Placement = (("volume", Named), ("pos", Named), ("rot", Named)),
    # ...
```

- Schema written simply as a static Python data structure
  - (link to `gegede.schema.Schema`)
- Follows naming and function prototype conventions of Geant4 geometry construction methods.
- Linear dimensions are usually taken as "half lengths" (eg, `dx`)
- Attributes defined as `(name, type)` pair. The type is either a Python type object or a string that evaluates to a `pint.Quantity` unit object and which provides a default value.
- Weakly typed references to other objects via `gegede.types.Named` type.

Exporters

Exporters produce persistent representations of GeGeDe objects.

Some export formats supported by GeGeDe are:

- **GDML** for Geant4. Uses `lxml.etree` for assuring valid XML.
- **ROOT** direct `TGeo` object creation (requires PyROOT)
- **JSON** trivial dump preserving GeGeDe internal object schema.
- **OIV** OpenInventor SceneGraph

New exporters should be easy to develop.

- Depends on the format, of course.
- Developing and testing the GDML exporter took about 2 hours.
- The JSON one took about 2 minutes!
  - :) It's a "cheat" as it just uses `json.dumps()`!

GeGeDe supports exporters as independent Python modules.

- Contributions back to GeGeDe are welcome!

Installation

GeGeDe requires:

- Python 2.7/3.5+
- Pint (for units)
- LXML (XML support)
- PyROOT (optional)

Install from source:

```
$ git clone https://github.com/brettviren/gegede.git
$ cd gegede/
$ python setup.py install
```

Install from PyPI:

```
$ pip install gegede
```

Interfaces

Command line:

```
$ gegede-cli -h
usage: gegede-cli [-h] [-w WORLD] [-f FORMAT] [-o OUTPUT]
                  [-V] [-O] [-F]
                  [-d DOT_FILE] [-D DOT_HIERARCHY]
                  config [config ...]
```

In principle can be used as module in a larger application by understanding:

```
gegede.main.main()
```

Documentation

Documentation starts with the main `README` on GitHub:

https://github.com/brettviren/gegede/

GeGeDe Status

- Main users now are doing DUNE Near Detector design studies.
  - They got up to speed using GeGeDe very quickly.
  - New users are welcome. You don't have to study neutrinos! :)
- Fairly stable code:
  - ✓ No open issues in the GitHub project!
    - ~ GDML export is best supported, others may have hiccups.
  - ✓ Recent contributions from DUNE (J. Palomino) adding more shapes
    - × Still need a few more for have 100% coverage of all Geant4 shapes!
  - ✓ Python 3.5+ support **just** added.
    - ~ Maybe some 2.7'isms still left uncovered.
- I'll continue to support GeGeDe at the level of bug fixes.
  - :) Major new features will likely be accepted but let's talk first.