SolidPython Documentation

Release 0.1.2

Evan Jones

Mar 10, 2023

Contents

Contents:

Hey! All the energy and improvements in this project are going into **SolidPython V2**. Check it out at Github or on its PyPI page before you commit to an older version.

CHAPTER 1

SolidPython

CHAPTER 2

SolidPython: OpenSCAD for Python

SolidPython is a generalization of Phillip Tiefenbacher's openscad module, found on Thingiverse. It generates valid OpenSCAD code from Python code with minimal overhead. Here's a simple example:

This Python code:

```
from solid import *
d = difference()(
    cube(10),
    sphere(15)
)
print(scad_render(d))
```

Generates this OpenSCAD code:

```
difference(){
    cube(10);
    sphere(15);
}
```

That doesn't seem like such a savings, but the following SolidPython code is a lot shorter (and I think clearer) than the SCAD code it compiles to:

```
from solid import *
from solid.utils import *
d = cube(5) + right(5)(sphere(5)) - cylinder(r=2, h=6)
```

Generates this OpenSCAD code:

```
difference(){
    union(){
        cube(5);
        translate( [5, 0,0]){
            sphere(5);
        }
```

(continues on next page)

(continued from previous page)

```
    }
    cylinder(r=2, h=6);
}
```

CHAPTER 3

Advantages

Because you're using Python, a lot of things are easy that would be hard or impossible in pure OpenSCAD. Among these are:

- built-in dictionary types
- mutable, slice-able list and string types
- recursion
- external libraries (images! 3D geometry! web-scraping! . . .)

CHAPTER 4

Installing SolidPython

- Install latest release via PyPI:

```
pip install solidpython
```

  (You may need to use `sudo pip install solidpython`, depending on your environment. This is commonly discouraged though. You'll be happiest working in a virtual environment where you can easily control dependencies for a given project)

- Install current master straight from Github:

```
pip install git+https://github.com/SolidCode/SolidPython.git
```

CHAPTER 5

Using SolidPython

- Include SolidPython at the top of your Python file:

```
from solid import *
from solid.utils import *  # Not required, but the utils module is useful
```

(See this issue for a discussion of other import styles)

- OpenSCAD uses curly-brace blocks ({}) to create its tree. SolidPython uses parentheses with comma-delimited lists.

OpenSCAD:

```
difference(){
    cube(10);
    sphere(15);
}
```

SolidPython:

```
d = difference()(
    cube(10),  # Note the comma between each element!
    sphere(15)
)
```

- Call `scad_render(py_scad_obj)` to generate SCAD code. This returns a string of valid OpenSCAD code.
- *or*: call `scad_render_to_file(py_scad_obj, filepath.scad)` to store that code in a file.
- If `filepath.scad` is open in the OpenSCAD IDE and Design => 'Automatic Reload and Compile' is checked in the OpenSCAD IDE, running `scad_render_to_file()` from Python will load the object in the IDE.
- Alternately, you could call OpenSCAD's command line and render straight to STL.

CHAPTER 6

Importing OpenSCAD code

- Use `solid.import_scad(path)` to import OpenSCAD code. Relative paths will

check the current location designated in OpenSCAD library directories.

Ex:

`scadfile.scad`

```
module box(w,h,d){
    cube([w,h,d]);
}
```

`your_file.py`

```
from solid import *

scadfile = import_scad('/path/to/scadfile.scad')
b = scadfile.box(2,4,6)
scad_render_to_file(b, 'out_file.scad')
```

- Recursively import OpenSCAD code by calling `import_scad()` with a directory argument.

```
from solid import *

# MCAD is OpenSCAD's most common utility library: https://github.com/openscad/MCAD
# If it's installed for OpenSCAD (on MacOS, at: ``$HOME/Documents/OpenSCAD/
↪libraries``)
mcad = import_scad('MCAD')

# MCAD contains about 15 separate packages, each included as its own namespace
print(dir(mcad)) # => ['bearing', 'bitmap', 'boxes', etc...]
mount = mcad.motors.stepper_motor_mount(nema_standard=17)
scad_render_to_file(mount, 'motor_mount_file.scad')
```

- OpenSCAD has the `use()` and `include()` statements for importing SCAD code, and SolidPython has them, too. They pollute the global namespace, though, and you may have better luck with `import_scad()`,

Ex:

scadfile.scad

```
module box(w,h,d){
    cube([w,h,d]);
}
```

your_file.py

```
from solid import *

# use() puts the module `box()` into the global namespace
use('/path/to/scadfile.scad')
b = box(2,4,6)
scad_render_to_file(b, 'out_file.scad')
```

CHAPTER 7

Example Code

The best way to learn how SolidPython works is to look at the included example code. If you've installed SolidPython, the following line of Python will print(the location of) the examples directory:

```
import os, solid; print(os.path.dirname(solid.__file__) + '/examples')
```

Or browse the example code on Github here

Adding your own code to the example file solid/examples/solidpython_template.py will make some of the setup easier.

CHAPTER 8

Extra syntactic sugar

8.1 Basic operators

Following Elmo Mäntynen's suggestion, SCAD objects override the basic operators + (union), - (difference), and * (intersection). So

```
c = cylinder(r=10, h=5) + cylinder(r=2, h=30)
```

is the same as:

```
c = union()(
    cylinder(r=10, h=5),
    cylinder(r=2, h=30)
)
```

Likewise:

```
c = cylinder(r=10, h=5)
c -= cylinder(r=2, h=30)
```

is the same as:

```
c = difference()(
    cylinder(r=10, h=5),
    cylinder(r=2, h=30)
)
```

8.2 First-class Negative Space (Holes)

OpenSCAD requires you to be very careful with the order in which you add or subtract objects. SolidPython's `hole()` function makes this process easier.

Consider making a joint where two pipes come together. In OpenSCAD you need to make two cylinders, union them, then make two smaller cylinders, union them, then subtract the smaller from the larger.

Using hole(), you can make a pipe, specify that its center should remain open, and then add two pipes together knowing that the central void area will stay empty no matter what other objects are added to that structure.

Example:

```
outer = cylinder(r=pipe_od, h=seg_length)
inner = cylinder(r=pipe_id, h=seg_length)
pipe_a = outer - hole()(inner)
```

Once you've made something a hole, eventually you'll want to put something, like a bolt, into it. To do this, we need to specify that there's a given 'part' with a hole and that other parts may occupy the space in that hole. This is done with the `part()` function.

See solid/examples/hole_example.py for the complete picture.

8.3 Animation

OpenSCAD has a special variable, `$t`, that can be used to animate motion. SolidPython can do this, too, using the special function `scad_render_animated_file()`.

See solid/examples/animation_example.py for more details.

CHAPTER 9

solid.utils

SolidPython includes a number of useful functions in solid/utils.py. Currently these include:

9.1 Directions: (up, down, left, right, forward, back) for arranging things:

```
up(10)(
    cylinder()
)
```

seems a lot clearer to me than:

```
translate( [0,0,10])(
    cylinder()
)
```

I took this from someone's SCAD work and have lost track of the original author.
My apologies.

9.2 Arcs

I've found this useful for fillets and rounds.

```
arc(rad=10, start_degrees=90, end_degrees=210)
```

draws an arc of radius 10 counterclockwise from 90 to 210 degrees.

```
arc_inverted(rad=10, start_degrees=0, end_degrees=90)
```

draws the portion of a 10x10 square NOT in a 90 degree circle of radius 10. This is the shape you need to add to make fillets or remove to make rounds.

9.3 Extrude Along Path

`solid.utils.extrude_along_path()` is quite powerful. It can do everything that OpenSCAD's `linear_extrude()` `` and ``rotate_extrude()` can do, and lots, lots more. Scale to custom values throughout the extrusion. Rotate smoothly through the entire extrusion or specify particular rotations for each step. Apply arbitrary transform functions to every point in the extrusion.

See solid/examples/path_extrude_example.py for use.

9.4 Bill Of Materials

Put `@bom_part()` before any method that defines a part, then call `bill_of_materials()` after the program is run, and all parts will be counted, priced and reported.

The example file solid/examples/bom_scad.py illustrates this. Check it out.

9.5 solid.screw_thread

solid.screw_thread includes a method, thread() that makes internal and external screw threads.

See solid/examples/screw_thread_example.py for more details.

9.6 solid.splines

solid.splines contains functions to generate smooth Catmull-Rom curves through control points.

```
from solid import translate
from solid.splines import catmull_rom_polygon, bezier_polygon
from euclid3 import Point2

points = [ Point2(0,0), Point2(1,1), Point2(2,1), Point2(2,-1) ]
shape = catmull_rom_polygon(points, show_controls=True)

bezier_shape = translate([3,0,0])(bezier_polygon(points, subdivisions=20))
```

See solid/examples/splines_example.py for more details and options.

9.7 Jupyter Renderer

Render SolidPython or OpenSCAD code in Jupyter notebooks using ViewSCAD, or install directly via:

```
pip install viewscad
```

(Take a look at the repo page, though, since there's a tiny bit more installation required)

CHAPTER 10

Contact

Enjoy, and please send any questions or bug reports to me at `evan_t_jones@mac.com`.

Cheers!

Evan

CHAPTER 11

License

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

Full text of the license.

Some class docstrings are derived from the OpenSCAD User Manual, so are available under the Creative Commons Attribution-ShareAlike License.

CHAPTER 12

Library Reference

CHAPTER 13

Indices and tables

- genindex
- modindex
- **search**

  members