How to Write Scripts for pypeVue

The pypeVue program is an interpreter that reads and translates scripts describing geometric figures. The program's output is constructive solid geometry code suitable for input to OpenSCAD. The last page of this document shows results from several scripts.

Layout and cylinder scripts tell pypeVue locations of posts to generate, and where to draw cylinders between posts. A script typically is read from a file, with a name specified by a command line parameter in the form"f=*filename*". When no file is specified, pypeVue uses a simple built-in script, and references pypeVue.codeBase.scad backend code in the output file it writes, pypeVue.scad.

In a script file, marker lines – lines starting with one of `=P`, `=L`, `=C`, or `=A` – denote various kinds of lines or sections: Parameter line, Layout section, Cylinder section, Arithmetic line. Comment lines include: blank lines; non-marker lines starting with `=`; and lines before the first marker line.

For examples of scripts, see files like eg-cap-5, eg-fatpost-redstar, eg-pentagon-script, eg-several-kinds, and eg-two-rings. For illustrations of output, see last page. For details about about arithmetic and about setting parameters, see the Arithmetic and Parameter Setting sections below. For directions on how to run pypeVue automatically when you change a script, see file README.

Layout Script Section

A Layout Script Section is a series of entries between an =L marker and [usually] an =C marker. =A and =P lines may appear within an =L section. There are several kinds of entries: Base point; Collection of points; Line of points; Polygon; Rectangle grid; and Triangle grid of points. Each entry begins with a code letter, B, C, I, L, O, P, R, or T to designate its type. The code letter is followed by several numeric parameters (either numbers or simple variables; see next section) and then is terminated by a semicolon. Examples: "P5,1,0;" "P5,#v,0;". Whenever an entry generates points, they are assigned post numbers in ascending order.

pypeVue applies a scale factor to all the location data (coordinates and radii) when producing output. The scale factor is given by parameter SF and its default value is 100.

Each **base point** entry has three parameters: *x,y,z,* which specify an anchor point, or base point, denoted *BP*, for use when treating C, L, P, R, or T entries. By default *BP* is (0,0,0). Example: "B 5.7, 2.2, 1.5;" sets *BP* to (5.7, 2.2, 1.5). Files eg-cap-5, eg-cap-6, and eg-several-kinds move the base point before making polygons, grids, and lines of points.

Each **polygon** entry has three parameters, *n, r, a0. n* is corner count; *r* is radius from *BP* to corners; and *a0* is the angle from *BP* to the first corner, in degrees CCW from the positive x axis. Post numbers increase going CCW. For example, "P5,1,0;" denotes a pentagon of posts with radius 1 (before scaling). As another example, "P12, 3.37, 11.5;" denotes a 12-gon of radius 3.37, rotated 11.5˚ CCW.

Each **rectangle grid** entry has four parameters, *r, c, dx, dy.* Integers *r* and *c* are the number of rows and columns to generate; decimals *dx* and *dy* are the width and height between columns and rows. Example: "R 5, 7, 2.2, 1.5;" makes a rectangular grid of posts, with five rows and seven columns, with columns spaced 2.2 apart and rows spaced 1.5 apart (before scaling). **Note: for rectangle and triangle grids**, points in the same row have consecutive numbers. A grid's first point, in row 0, is at *BP*+(0,0,0), the second is at *BP*+(dx,0,0), and so forth. Row 1 starts at *BP*+(0,dy,0), then *BP*+(dx,dy,0) for a **rectangle grid**, or at *BP*+(-dx/2,dy,0) , then *BP*+(dx/2,dy,0) for a **triangle grid**. Row 2 starts at *BP*+(0,2·dy,0), then

BP+(dx,2·dy,0), then *BP*+(2·dx,2·dy,0), etc.

Each **triangle grid** entry has four parameters, *r, c, dx, dy,* where *r* and *c* are the number of rows and columns to generate and *dx*, *dy* are width and height between columns and rows. Even-numbered rows have one fewer points than do odd-numbered rows. Thus, row 0 has *r* posts in it, row 1 has *r*+1, row 2 has *r,* and so forth, giving *rc*+(*r*//2) points in all. Example: "T5, 5, 1, 0.866" produces 27 posts (27=5·5+5//2) that are on a grid of equilateral triangles with unit side. As noted in the previous paragraph, even-numbered rows are offset by *dx*/2 relative to odd-numbered rows.

Each **line of points** entry has four parameters: *n, dx, dy, dz.* *n* is the number of points in the line. The first point in the line is at *BP* and (*dx,dy,dz*) is the x,y,z step between consecutive points. Example: If *BP* is (2, -2, .2), then `L4 3,1,.01;` makes posts at (2, -2, 0.2); (5, -1, 0.21); (8, 0, 0.22); (11, 1, 0.23).

Each **collection of points** entry has a list of x,y,z values, and a semicolon terminator. For each x,y,z value, a post is produced at *BP*+(x,y,z). Example: If BP is (0.1, 0.2, 0.3), then `C 1,1,1, 4,4,1, 5,6,7 7,11,13;` makes posts at (1.1, 1.2, 1.3); (4.1, 4.2, 1.3); (5.1, 6.2, 7.3); and (7.1, 11.2, 13.3), in order.

Origin point, OP: The command parameter postAxial and the **origin point** layout operator **O** control the alignment of center-lines of posts. • When postAxial is true, center-lines align with the z axis. • When postAxial is false, center-lines radiate from the origin point. An origin point entry has three parameters: *x,y,z,* specifying a point. Example: suppose postAxial is false and O0,0,-4 and C 4,0,0 are given. In this case, the post with base at (4,0,0) tilts away from the z axis at a 45° angle. *Note:* Only the last-specified OP has effect. Specifying OP more than once is not useful.

Arithmetic

An arithmetic line contains Python code. During script processing, the code will execute in a contained environment. At the most basic level, the code can create, reference, and change simple variables – variables with single-letter names. The =C and =L sections can refer to simple variables by a hash mark, #, followed by a variable name, a to z or A to Z. In an =C section, any #v appearance fetches the value of variable v. In an =L section, a #v appearance in a list of numbers (ie after an action code, or before a semicolon) fetches the value of variable v, while an appearance after a semicolon, or before an action code, sets its value equal to the next available post number. The following example script prints out (twice) "`p q r s t = 8 9 8 6 14`" and produces a green-ringed octagon and a red-ringed hexagon. The output prints twice because pypeVue goes through the whole script twice, once when making posts and once when making cylinders.

```
=A p=8; s=6
=P postDiam=.1
=L P#p,1,0;  #r  P#s,2,0;  #t
=A q=p+1; print ('p q r s t =',p,q,r,s,t)
=C G0,1;;;;;;;0; R#p#q;;;;;;;#p;
```

More-involved code in an arithmetic line can access pypeVue's data structures, import code to compute coordinates and connections, etc. How to do so will be documented in future. Error handling also will be added; at the moment, python syntax errors produce a messy Python traceback and message.

Cylinder Script Section

A Cylinder Script Section is a series of entries between an =C marker and [usually] an =L marker or end of file. =A and =P lines may appear within an =C section. Each entry, specifying one cylinder, is composed of optional elements <color>, <diam>, <post>, and <level> in any order. It is completed by a semicolon, as explained below. Here is an

example script which draws four thin differently-colored horizontal cylinders at different levels: "qG 0a1a; R2bb3; C4cc5; M6dd7;". In this example, "R2bb3;" specifies a red cylinder of thickness q from level b of post 2 to level b of post 3.

<color> is one of G, Y, R, B, C, M, or W for green, yellow, red, blue, cyan, magenta, white.

<diam> is a letter, p, q, r, … w. By default, pDiam=0.06 and qDiam=0.02. The thicknesses of r, s,...w scale geometrically according to parameter dRatio, which is sqrt(2) by default. Thus, r is about 1.41·q, s is 2·q, and so forth. File eg-two-rings (see illustrations) makes dRatio smaller, for more-gradual increase in pipe size.

Each cylinder entry allows optional <post> numbers and optional <level> codes, that is, a letter (a-e) for a level on a post. Level a is low, e is high, and others are spread proportionally between. For example, "14b26c;" specifies a cylinder from level b on post 14 to level c on post 26, using whatever color and diameter were previously specified. As another example, "7e;" specifies a cylinder to level e on post 7, from the most recently given post and level, using previously specified color and diameter.

The semicolon after a cylinders entry is required punctuation. When a semicolon appears, a cylinder is produced. Ordinarily, the cylinder uses the most recent post numbers, levels, color, and diameter. But if the entry just concluded had no post numbers in it, a special action occurs: the latest post numbers are each incremented by 1 before the cylinder is drawn. This allows writing "1,11;;;;;" instead of "1,11; 2,12; 3,13; 4,14; 5,15;", or allows writing "1,2;;;;1;" instead of "1,2;3;4;5;1;", which in turn is equivalent to "1,2; 2,3; 3,4; 4,5; 5,1;".

A special character "/" also is allowed in cylinders entries. The slash is shorthand for swapping the two most recent levels. This allows writing, for example, "ae1,2;/;/;/;/1;" instead of "ae1,2;a3;e4;a5;e1;".

Most elements are optional within each entry, and elements not given remain as previously specified. For example, the specification "Ccc0,1;2;3;4;" draws four cyan-colored cylinders, in a chain from post 0 to 1 to 2 to 3 to 4, with all cylinder-ends at level c. As another example, "qCab0,1;c;d;e;" draws four thin cylinders between posts 0 and 1, going from 0a to 1b; from 0b to 1c; from 0c to 1d; and from 0d to 1e.

Initial defaults for color, thickness, level, and post are G, p, a, 0. White space and characters not listed above are ignored, except that they delimit post numbers. For example, the entries quoted in the following set are equivalent: { "Gp0a1e;" "pG0ae1;" "ae0Gp1;" }

Parameter Setting

Parameter values control the overall appearance of pypeVue output. These parameters include, among others: cylList, cylSegments, dRatio, endGap, f, pDiam, paramTxt, postDiam, postHi, postLabel, postList, qDiam, scadFile, and SF.

Parameters are given new values by entries like "pDiam=.1" or "postList=Y" on the command-line or in the =P section of a script file. The name must exactly match a parameter name as in the list above or table below. Else, an error message will print. Values should be properly formatted decimal integers for cylSegments and SF. Other numeric values must be properly formatted decimal numbers.

Note, in a list of parameter settings, each entry should be separated from other entries by white space, and should contain no white space itself.

General effects of parameters are shown in Table 1, next page. Here are a few details for selected parameters:

For **cylList** and **postList** boolean values, an initial character among f, F, N, or n denotes False, and anything else means True. Example: If parameter settings include "postList=t", a list of post coordinates will print out.

For **postLabel**, you can either indicate false – eg, "postLabel=f",to suppress post labels – or can specify color, size, and level codes to use when drawing numeric labels next to posts. Use the same codes for post-number color/size/level as for cylinder color/size/level: characters from 'GYRBCMW pqrstuvw abcde'. The default value, 'Bte', indicates blue numbers, 2.8 (or, √8) times as wide as qDiam, at top of post.

autoList, autoMax: These control automatic generation of edges and listing of such edges. When autoMax exceeds the length of some edges that could have been specified but weren't, those edges will be generated automatically, and will be listed in output if autoList is true. (With default values 0 and True, no edges will be generated, but would be listed if they were.) See eg-freq-6-auto for examples and more discussion. This mechanism is a low-grade substitute for Delaunay triangulation; for typical geodesics, it works adequately.

postAxial: When it is true, this constructs posts parallel to the z axis. Else, posts point out radially from a specified origin point (see: **O** layout operator).

codeBase, userCode, userPar0, userPar1, userPar2 – Via the first two of these parameters, you can specify files of SCAD code that control how OpenSCAD displays posts, pipes, and labels. The latter three, if specified, give the SCAD code some user parameter values; see examples in eg-two-rings, eg-two-helper.scad, eg-auto-test-2, eg-auto-help-2.scad. **codeBase** defaults to "pypeVue.codeBase.scad", the name of a file with basic post/pipes/label modules. See that file for further comments on the code structure. See a generated "pypeVue.scad" file for examples of calls to onePost(), oneLabel(), and oneCyl() modules. **userCode** defaults to "", that is, to an empty string, which tells pypeVue that no additional SCAD code is to be used. If you change **userCode**, make it the name of a reachable file. Note, modules defined in the **userCode** file preempt those of the same name in the **codeBase** file. See files eg-two-rings and eg-two-helper.scad for examples of substitute onePost() and oneCyl() modules, or see eg-auto-test-2 and eg-auto-help-2.scad for an example addOn() module.

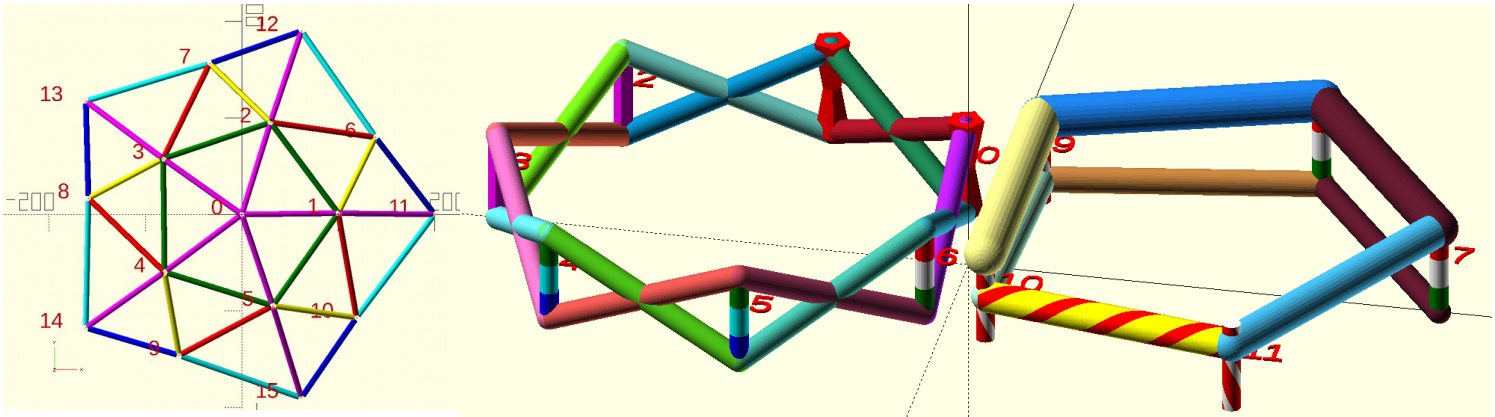
zSpread, zSize – These control a conformal mapping option that can flatten or spread out the skirts of a spherical or hemispherical figure, for easier viewing from above. See eg-two-rings for notes and examples. When zSpread is true, x and y are multiplied by 2/(1+z/zSize) before use. If figure's origin and radius are (0,0,0) and zSize, then x and y of equatorial points (where z is 0) are expanded by a factor of 2. Note, one could instead freely transform coordinates within code modules onePost(), oneLabel(), and oneCyl(), at the cost of also recomputing angles.

paramTxt technical note: • Initially, parameter values are set by python statements in the program. • Next, parameter value entries, each in a "var=value" form, are collected together into paramTxt, which is used from time to time to set any parameter values it specifies. • If there is a command-line entry f=*filename*, then parameter and script texts are read from the named file, and any parameters specified there are set when posts and cylinders are generated. • If a parameter is specified in both places (command-line and file), the **command-line value prevails over its file value**, unless the file gives **paramTxt** a blank value.

Table 1: General effects of Parameters

Parameter Name	Default or Initial Value	Parameter Use
autoList	True	yes/no: list auto-generated edges
autoMax	0	max length for auto-generated edges
cylList	False	yes/no for cylinders data listing
cylSegments	30	Number of sides/cylinder to generate
dRatio	sqrt(2)	Geometric ratio for q, r, s... cylinder diameters
endGap	0.03	Spacing (under/over) between cyl. ends and posts
f	'' (empty string)	Input script file name
pDiam	0.06	Default diameter for cylinders
paramTxt	'' (empty string)	Concatenation of command-line parameters
postAxial	True	yes/no: posts z-axis aligned, vs radial
postDiam	qDiam	Diameter for posts
postHi	0.16	Height of posts
postLabel	'Bte'	yes/no or color/size/level for post labeling
postList	False	yes/no for post coordinates listing
qDiam	0.02	alternate diameter for cylinders
scadFile	'pypeVue0.scad'	Output file name
SF	100	Overall scale factor
codeBase		
userCode,	'' (empty string)	
userPar0, 1, 2	'""' (quoted empty string)	Three string values for inputs to SCAD code; values also appear in Customizer panel
zSpread	False	yes/no for conformal mapping option
zSize	1	Conformal mapping distance control.

Illustrations: Upper, eg-cap-5 and eg-two-rings. Middle, eg-two-rings detail. Lower, eg-fatpost-redstar and a variant of eg-cap-5-freq-6-full-weave-hemisphere.



In this eg-two-rings example, userPar0 is 0.0. If you open the Customizer panel in OpenSCAD and change its value, then press F5 or enter, the colors will change because userPar0 is used in the color hash function in file eg-two-helper.scad.