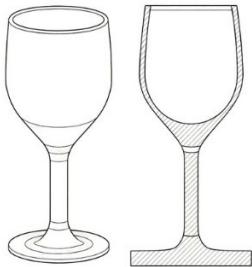


Comparing the workflow with traditional ones

(Example 1: Modeling a wine glass)

To illustrate the difference in workflow efficiency, consider the modeling of a standard wine glass. The underlying conceptual shape is shown below.



In a traditional script-based CAD workflow, the user must translate this visual concept directly into code. The following OpenSCAD script is a representative example of this approach.

```
1. // parameters
2.
3. detail = 100;      // mesh resolution
4. wall   = 2.4;     // wall thickness
5.
6. // base (foot)
7. base_d   = 75;
8. base_h   = 10;
9. base_edge = 0;    // 0-180°
10.
11. // stem
12. stem_d   = 8;
13. fillet_r   = 18;  // radius of foot-stem fillet
14. fillet_shape = 1; // 0.3 flat, 1 round, 2.5 steep
15.
16. // body (torso + bowl)
17. body_z      = 120; // sphere center height
18. body_d      = 70;  // body diameter
19. body_tilt   = 40;  // max sidewall angle
20. neck_d      = 50;  // transition diameter
```

```

21. body_top_scale = 3.5; // z-scale of upper body
22. glass_max_h     = 170; // max height
23. rim_th          = 0;   // rim thickness (0 = off)
24.
25. // derived values
26.
27. stem_offset = body_d/2 * cos(body_tilt) - stem_d/2;
28. stem_len    = stem_offset / sin(body_tilt);
29. stem_h      = body_z
30.           - (body_d/2) * sin(body_tilt)
31.           - stem_len * cos(body_tilt);
32.
33. inner_ball_d = stem_d - 2*wall;
34. inner_offset = (body_d/2 - wall)*cos(body_tilt) - stem_d/2 + wall;
35. inner_len    = inner_offset / sin(body_tilt);
36. inner_ball_z = body_z
37.           - (body_d - 2*wall)/2 * sin(body_tilt)
38.           - inner_len * cos(body_tilt)
39.           + wall * sin(body_tilt);
40.
41. inner_neck_z = body_z - (body_d - 2*wall)/2 * sin(body_tilt);
42. inner_neck_d = (body_d - 2*wall) * cos(body_tilt);
43.
44. neck_len = neck_d * sin(body_tilt);
45.
46. rim_h_aux  = body_d/2 - wall/4 - (glass_max_h - body_z)/body_top_scale;
47. rim_radius = sqrt((body_d - wall)*rim_h_aux - rim_h_aux*rim_h_aux);
48.
49.
50. // base / foot
51.
52. module base_part() {
53.     if (base_edge <= 90) {
54.         // center disk
55.         cylinder(h = base_h/2 + base_h/2 * sin(90 - base_edge),
56.                   d = base_d, $fn = detail);

```

```

57.
58.      // curved outer edge
59.      intersection() {
60.          cylinder(h = base_h, d = base_d, $fn = detail);
61.
62.          hull() {
63.              rotate_extrude(angle = 360, $fn = detail) {
64.                  translate([
65.                      base_d/2 - base_h/2 * cos(90 - base_edge),
66.                      base_h/2
67.                  ])
68.                  circle(d = base_h, $fn = detail);
69.              }
70.          }
71.      }
72.  } else {
73.      // curved base
74.      hull() {
75.          rotate_extrude(angle = 360, $fn = detail) {
76.              translate([base_d/2 - base_h/2, base_h/2])
77.              circle(d = base_h, $fn = detail);
78.          }
79.      }
80.
81.      // center disk
82.      cylinder(h = base_h/2,
83.                  d = base_d - base_h * cos(180 - base_edge),
84.                  $fn = detail);
85.  }
86. }
87.
88.
89. // straight stem with inner ball cut-out
90.
91. module stem_part() {
92.     difference() {

```

```

93.     translate([0,0,base_h])
94.         cylinder(h = stem_h - base_h + wall * sin(body_tilt),
95.                     d = stem_d, $fn = detail);
96.
97.     hull() {
98.         translate([0,0,inner_ball_z])
99.             sphere(d = inner_ball_d, $fn = detail);
100.
101.        translate([0,0,inner_neck_z])
102.            cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
103.    }
104. }
105. }
106.
107.
108. // fillet from base to stem
109.
110. module base_stem_fillet() {
111.     difference() {
112.         translate([0,0,base_h])
113.             cylinder(h = fillet_r * fillet_shape,
114.                         d = stem_d + 2*fillet_r, $fn = detail);
115.
116.         rotate_extrude(angle = 360, $fn = detail) {
117.             translate([stem_d/2 + fillet_r, base_h + fillet_r * fillet_shape])
118.                 scale([1, fillet_shape])
119.                     circle(r = fillet_r, $fn = detail);
120.         }
121.     }
122. }
123.
124.
125. // lower body (torso bottom)
126.
127. module lower_body_part() {
128.     difference() {

```

```

129.     translate([0,0,body_z])
130.         sphere(d = body_d, $fn = detail);
131.
132.     translate([0,0,body_z])
133.         sphere(d = body_d - 2*wall, $fn = detail);
134.
135.     // remove upper half of the sphere
136.     translate([-body_d, -body_d, body_z])
137.         cube([2*body_d, 2*body_d, body_d]);
138.
139.     // bottom cut (angle limit)
140.     translate([-body_d,
141.                 -body_d,
142.                     body_z - body_d - body_d/2 * sin(body_tilt)])
143.         cube([2*body_d, 2*body_d, body_d]);
144.
145.     // blend to inner stem ball
146.     hull() {
147.         translate([0,0,inner_ball_z])
148.             sphere(d = inner_ball_d, $fn = detail);
149.
150.         translate([0,0,inner_neck_z])
151.             cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
152.     }
153. }
154. }
155.
156.
157. // transition from stem to lower body
158.
159. module stem_body_transition() {
160.     difference() {
161.         rotate_extrude(angle = 360, $fn = detail) {
162.             translate([-stem_d/2, stem_h])
163.                 rotate(body_tilt)
164.                     square([wall, stem_len + 0.1]);

```

```

165.     }
166.
167.     hull() {
168.         translate([0,0,inner_ball_z])
169.             sphere(d = inner_ball_d, $fn = detail);
170.
171.         translate([0,0,inner_neck_z])
172.             cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
173.     }
174. }
175. }
176.
177.
178. // rounded neck between stem and body
179.
180. module rounded_neck_part() {
181.     difference() {
182.         translate([0,0,stem_h - neck_len/4])
183.             cylinder(h = neck_len/2,
184.                     d = stem_d + neck_d,
185.                     $fn = detail);
186.
187.         rotate_extrude(angle = 360, $fn = detail) {
188.             translate([
189.                 stem_d/2 + neck_d/2,
190.                 stem_h - neck_d/2 * cos((180 - body_tilt)/2)
191.             ])
192.             circle(d = neck_d * 1.002, $fn = detail);
193.         }
194.
195.         hull() {
196.             translate([0,0,inner_ball_z])
197.                 sphere(d = inner_ball_d, $fn = detail);
198.
199.             translate([0,0,inner_neck_z])
200.                 cylinder(h = 0.1, d = inner_neck_d, $fn = detail);

```

```

201.      }
202.
203.      translate([0,0,body_z])
204.          sphere(d = body_d - 2*wall, $fn = detail);
205.    }
206.  }
207.
208.
209. // upper body / bowl
210.
211. module upper_body_part() {
212.     difference() {
213.         translate([0,0,body_z])
214.             scale([1,1,body_top_scale])
215.                 sphere(d = body_d, $fn = detail);
216.
217.         // top cut at max height
218.         translate([0,0,glass_max_h + 50*body_top_scale - rim_th/2])
219.             cube(100*body_top_scale, center = true);
220.
221.         // bottom cut near rim zone
222.         translate([0,0,-50*body_top_scale + body_z - rim_th/2])
223.             cube(100*body_top_scale, center = true);
224.
225.         translate([0,0,body_z])
226.             scale([1,1,body_top_scale])
227.                 sphere(d = body_d - 2*wall, $fn = detail);
228.     }
229. }
230.
231.
232. // rim (no output if rim_th <= 0)
233.
234. module rim_part() {
235.     if (rim_th > 0) {
236.         rotate_extrude(angle = 360, $fn = detail) {

```

```

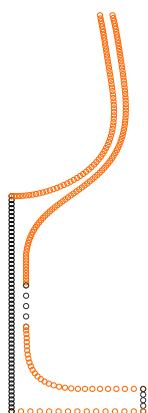
237.         translate([rim_radius, glass_max_h - rim_th/2])
238.             circle(d = rim_th, $fn = detail);
239.     }
240. }
241. }
242.
243.
244. // assembly
245.
246. module goblet() {
247.     union() {
248.         base_part();
249.         stem_part();
250.         base_stem_fillet();
251.         lower_body_part();
252.         stem_body_transition();
253.         rounded_neck_part();
254.         upper_body_part();
255.         rim_part();
256.     }
257. }
258.
259. goblet();

```

Note that the above script is rearranged from the work found here: “Simple Wine Glass Generator OpenSCAD by RacoonX on Thingiverse, URL: <https://www.thingiverse.com/thing:4914266>. ” Rendering the above script models the wine glass follows.



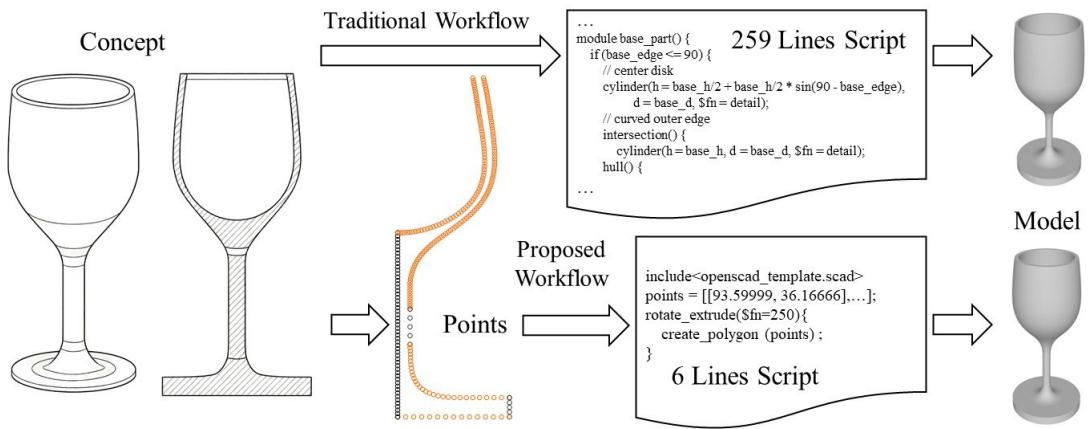
In contrast, the proposed workflow separates geometric definition from the coding process. In this workflow, the user first generates the relevant point clouds using the IPCM systems. (For detailed operational instructions, please refer to the documentation available at: <https://commons-repo.github.io/002-research/>). Subsequently, a concise script is written based on the developed template and functions, which are also available at the aforementioned URL. The following figures illustrate the generated points, the corresponding script, and the final rendered model, respectively.



```
1. include<openscad_template.scad>
2. points = [[93.59999, 36.16666], [94.24068, 39.8713], ..., [82, 36.16666]];
3.
4. rotate_extrude($fn=250){
5.     create_polygon (points) ;
6. }
7.
```



The following figure compares the abovementioned workflows.



As seen in the above figure, the contrast between the workflows provides empirical evidence of the cognitive and computational difficulty inherent in the traditional workflow. The traditional workflow imposes a significant burden by forcing the user to bridge the gap between visual intent and rigid code: the user must not only define coordinates but also manually implement the heavy geometric calculations required for curve evaluation and Boolean operations. This results in a verbose script of approximately 259 lines, introducing strict syntax requirements and debugging demands that distract from the creative process. In contrast, the proposed framework systematically abstracts this complexity into the IPCM Layer, reducing the script required from the user to a mere 6 lines. This represents a code reduction of over 97%, effectively eliminating the cognitive and computational barrier and allowing the user to focus entirely on the design intent rather than algorithmic implementation.