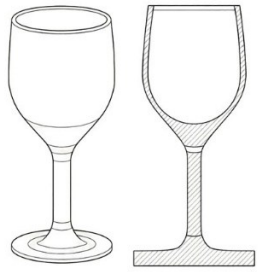
Comparing the workflow with traditional approaches

(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.

```
// parameters

detail = 100;     // mesh resolution
wall   = 2.4;    // wall thickness

// base (foot)
base_d    = 75;
base_h    = 10;
base_edge = 0;   // 0–180°

// stem
stem_d       = 8;
fillet_r     = 18;  // radius of foot–stem fillet
fillet_shape = 1;   // 0.3 flat, 1 round, 2.5 steep

// body (torso + bowl)
body_z         = 120;  // sphere center height
body_d         = 70;   // body diameter
body_tilt      = 40;   // max sidewall angle
neck_d         = 50;   // transition diameter
```

```
body_top_scale = 3.5;  // z-scale of upper body
glass_max_h    = 170;  // max height
rim_th         = 0;    // rim thickness (0 = off)

// derived values

stem_offset = body_d/2 * cos(body_tilt) - stem_d/2;
stem_len    = stem_offset / sin(body_tilt);
stem_h      = body_z
            - (body_d/2) * sin(body_tilt)
            - stem_len * cos(body_tilt);

inner_ball_d = stem_d - 2*wall;
inner_offset = (body_d/2 - wall)*cos(body_tilt) - stem_d/2 + wall;
inner_len    = inner_offset / sin(body_tilt);
inner_ball_z = body_z
             - (body_d - 2*wall)/2 * sin(body_tilt)
             - inner_len * cos(body_tilt)
             + wall * sin(body_tilt);

inner_neck_z = body_z - (body_d - 2*wall)/2 * sin(body_tilt);
inner_neck_d = (body_d - 2*wall) * cos(body_tilt);

neck_len = neck_d * sin(body_tilt);

rim_h_aux  = body_d/2 - wall/4 - (glass_max_h - body_z)/body_top_scale;
rim_radius = sqrt((body_d - wall)*rim_h_aux - rim_h_aux*rim_h_aux);


// base / foot

module base_part() {
    if (base_edge <= 90) {
        // center disk
        cylinder(h = base_h/2 + base_h/2 * sin(90 - base_edge),
                 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() {
```

```
        translate([0,0,base_h])
            cylinder(h = stem_h - base_h + wall * sin(body_tilt),
                    d = stem_d, $fn = detail);

        hull() {
            translate([0,0,inner_ball_z])
                sphere(d = inner_ball_d, $fn = detail);

            translate([0,0,inner_neck_z])
                cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
        }
    }
}


// fillet from base to stem

module base_stem_fillet() {
    difference() {
        translate([0,0,base_h])
            cylinder(h = fillet_r * fillet_shape,
                    d = stem_d + 2*fillet_r, $fn = detail);

        rotate_extrude(angle = 360, $fn = detail) {
            translate([stem_d/2 + fillet_r, base_h + fillet_r * fillet_shape])
                scale([1, fillet_shape])
                    circle(r = fillet_r, $fn = detail);
        }
    }
}


// lower body (torso bottom)

module lower_body_part() {
    difference() {
```

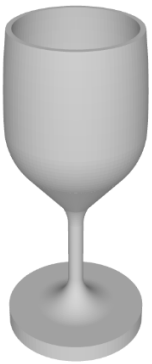
```
        translate([0,0,body_z])
            sphere(d = body_d, $fn = detail);

        translate([0,0,body_z])
            sphere(d = body_d - 2*wall, $fn = detail);

        // remove upper half of the sphere
        translate([-body_d, -body_d, body_z])
            cube([2*body_d, 2*body_d, body_d]);

        // bottom cut (angle limit)
        translate([-body_d,
                   -body_d,
                   body_z - body_d - body_d/2 * sin(body_tilt)])
            cube([2*body_d, 2*body_d, body_d]);

        // blend to inner stem ball
        hull() {
            translate([0,0,inner_ball_z])
                sphere(d = inner_ball_d, $fn = detail);

            translate([0,0,inner_neck_z])
                cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
        }
    }
}


// transition from stem to lower body

module stem_body_transition() {
    difference() {
        rotate_extrude(angle = 360, $fn = detail) {
            translate([-stem_d/2, stem_h])
                rotate(body_tilt)
                    square([wall, stem_len + 0.1]);
```

```
        }

        hull() {
            translate([0,0,inner_ball_z])
                sphere(d = inner_ball_d, $fn = detail);

            translate([0,0,inner_neck_z])
                cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
        }
    }
}


// rounded neck between stem and body

module rounded_neck_part() {
    difference() {
        translate([0,0,stem_h - neck_len/4])
            cylinder(h = neck_len/2,
                    d = stem_d + neck_d,
                    $fn = detail);

        rotate_extrude(angle = 360, $fn = detail) {
            translate([
                stem_d/2 + neck_d/2,
                stem_h - neck_d/2 * cos((180 - body_tilt)/2)
            ])
                circle(d = neck_d * 1.002, $fn = detail);
        }

        hull() {
            translate([0,0,inner_ball_z])
                sphere(d = inner_ball_d, $fn = detail);

            translate([0,0,inner_neck_z])
                cylinder(h = 0.1, d = inner_neck_d, $fn = detail);
```

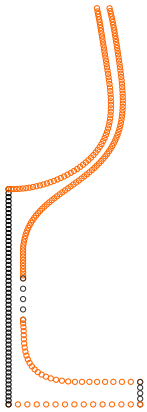
```
        }

        translate([0,0,body_z])
            sphere(d = body_d - 2*wall, $fn = detail);
    }
}


// upper body / bowl

module upper_body_part() {
    difference() {
        translate([0,0,body_z])
            scale([1,1,body_top_scale])
                sphere(d = body_d, $fn = detail);

        // top cut at max height
        translate([0,0,glass_max_h + 50*body_top_scale - rim_th/2])
            cube(100*body_top_scale, center = true);

        // bottom cut near rim zone
        translate([0,0,-50*body_top_scale + body_z - rim_th/2])
            cube(100*body_top_scale, center = true);

        translate([0,0,body_z])
            scale([1,1,body_top_scale])
                sphere(d = body_d - 2*wall, $fn = detail);
    }
}


// rim (no output if rim_th <= 0)

module rim_part() {
    if (rim_th > 0) {
        rotate_extrude(angle = 360, $fn = detail) {
```

```
            translate([rim_radius, glass_max_h - rim_th/2])
                circle(d = rim_th, $fn = detail);
        }
    }
}


// assembly

module goblet() {
    union() {
        base_part();
        stem_part();
        base_stem_fillet();
        lower_body_part();
        stem_body_transition();
        rounded_neck_part();
        upper_body_part();
        rim_part();
    }
}

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.

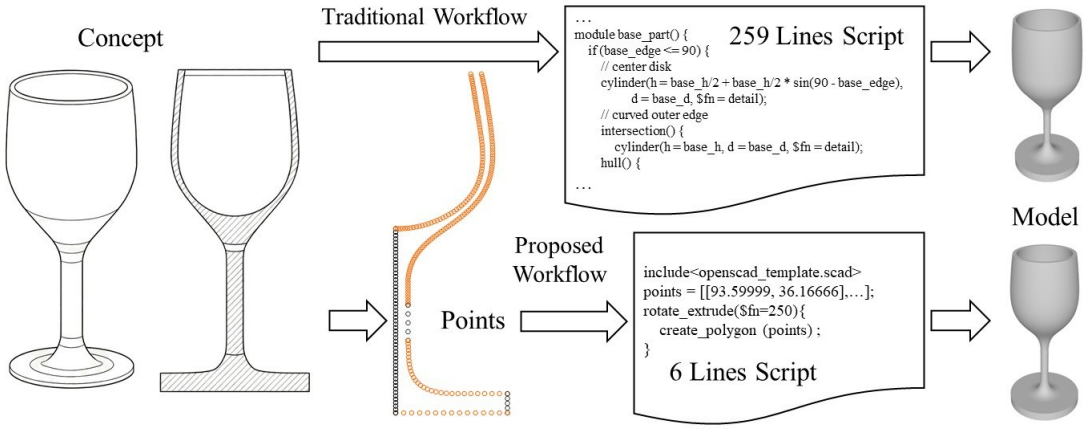


In contrast, the proposed workflow separates geometric definition from the coding process. In this workflow, the user first generates the relevant point clouds using the IPCM systems. (For detailed operational instructions, please refer to the documentation available at: https://commons-repo.github.io/002-research/). 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.



```
include<openscad_template.scad>
points = [[93.59999, 36.16666], [94.24068, 39.8713], …, [82, 36.16666]];

rotate_extrude($fn=250){
    create_polygon (points) ;
}

```

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.