Flower Keychain Case Study

Case Study: Extending the System — A Parametric Flower Keychain

Project: Davenport Designs — flower keychain with initial, boy/girl font option **Tools:** OpenSCAD, Google Fonts, Terminal/Bash, VS Code, Claude **Outcome:** A second fully-owned parametric generator, reusing the name-plate system's core techniques, exporting 3 independently colorable parts

---

The Problem

With the name-plate generator running smoothly, the next request was a second product: a flower-shaped keychain with a single initial in the center, available in a boy style and a girl style, printed in up to three colors. The requirement was the same as the first build — full ownership of the geometry, no MakerWorld license, no per-design fees — but the shape was entirely new, and the color requirement was more specific this time: not just two parts, but three (background petals, a center "button," and the letter), each independently assignable to its own filament.

The Plan

Rather than starting over, this build reused every proven piece of the name-plate system: OpenSCAD as the modeling engine, Google Fonts for the lettering (with the same "leave a placeholder, swap in a real font once installed" safeguard learned from the first build), inches as the only unit the script ever asks for, and a `part` variable that lets one script export multiple independently-colorable STL files instead of a single fused object.

Designing the Petals

Getting the petal shape right took three real iterations:

1. **A stretched ellipse.** The first attempt built each petal as a circle scaled long in one direction. At a small size it looked fine; once the

flower was resized larger, the thin, sharply-tapering tips read as a gear, not a flower.

2. **A “bubble” hull between two circles.** The second attempt hulled a small circle near the center with a larger one at the tip, giving a rounded, plump petal with no sharp point. Closer, but still not quite right — and by this point, guessing at the exact look from description alone had reached its limit.
3. **Reference-photo matching.** A photo of the actual target design changed the approach entirely. The real shape wasn’t a hull between two circles at all — it was several overlapping circles, one per petal, placed around a center point with enough overlap to merge into one continuous, gap-free scalloped edge. Rebuilding the petal module around that technique (plain circles, generous overlap, no separate hub) matched the reference almost immediately.

This is a pattern worth naming: for the name plate, the design goal was described in words and iterated through several rounds. For the flower, one reference photo did more than several rounds of verbal back-and-forth — worth reaching for early on future shapes, not as a last resort.

The Middle Layer: Reused, Then Rotated, Then Fixed

The center “button” started as a plain circle, but the reference photo showed something more specific: a smaller version of the *same* flower shape, sitting on top, rotated so its petals peek out between the background’s petals rather than sitting directly on top of them. Because the petal geometry was already its own reusable module, this was a small change — call the same shape-building function at a smaller scale with an added rotation offset, rather than building a second, different shape.

One bug slipped through this step: the background had a “closing” pass (grow, then shrink back) that guarantees it prints as one fused, gap-free solid — but the middle layer’s version of that same call was missed when it was added, so the middle printed as a set of technically-overlapping but visually distinct circles instead of one smooth shape. Same fix as the background, just applied to the layer that had been skipped.

The Hole: A Real Design Tension, Solved by Removing the Cleverness

The keychain hole caused more back-and-forth than anything else in this build, and it's worth documenting honestly because the resolution matters more than the individual fixes:

- First attempt: position the hole mathematically, centered on the background's own top petal.
- A screenshot suggested this was wrong — but the screenshot turned out to be a rotated 3D preview angle, not an actual geometry bug. A second, straight-down screenshot confirmed the original math was correct all along.
- The real complaint, once clarified, was about a different relationship entirely: the hole looked disconnected from the middle layer's petals, since the middle is intentionally rotated to interleave with the background — meaning the hole (aligned to a background petal) necessarily sits in a *valley* of the middle layer.
- The fix was changed to align the hole with a middle petal instead. Then reversed back to the background petal. Both are structurally valid — the two layers are, by design, never pointing the same direction at once, so no formula can satisfy both simultaneously.

The actual resolution wasn't a smarter formula. It was removing the formula entirely: two plain numbers, `hole_x_in` and `hole_y_in`, directly editable, no geometry math attached. Once position was just a number to move by hand rather than a value derived from petal angles, the back-and-forth stopped immediately. The lesson generalizes: when a parametric relationship has two valid, mutually exclusive interpretations, the right move is often to stop trying to infer the "correct" one and just expose a manual control.

Getting to a One-Paste Workflow

The name-plate project had already worked through the worst of the Mac-specific friction (silent GUI export failures, a non-standard OpenSCAD install location, PATH issues). This build hit one more: browser downloads of `.sh` script files were unreliable, and pasting a heredoc block to create one directly in Terminal left the terminal stuck mid-paste on more than one occasion.

The fix was to stop generating a separate script file at all. Instead, a single self-contained command does everything in one paste: it searches the whole home folder for the `.scad` file by name, `cd`s into wherever it's actually saved, runs OpenSCAD once per part, and reports the output location — with no file to download, no permissions to set, and no folder-navigation required beforehand. This is now the standing pattern for both generators.

Takeaways

- **A reference photo beats several rounds of verbal description.** The petal shape only converged once an actual image was available to match against.
- **Reused infrastructure pays off immediately.** Every hard-won piece of the name-plate system — inches-only sizing, the closing-pass solidity trick, the multi-part export pattern — transferred directly to a completely different shape with no rework.
- **Not every bug is a bug.** One full round of “the hole is wrong” turned out to be a rendering camera angle, not the model. A single top-down screenshot resolved it in one exchange.
- **When two correct answers conflict, stop calculating and expose a dial.** The hole-position saga only ended once the script stopped trying to be clever about where the hole “should” go and simply asked for a number.
- **Simplify the delivery mechanism, not just the design.** The final workflow improvement wasn’t a script feature at all — it was removing the script file as a concept and replacing it with one portable, self-locating command.

Where This Leaves the Business

Davenport Designs now has two fully-owned parametric generators sharing one underlying toolkit and one workflow: edit a value, paste one command, get color-ready parts. The second product line took a fraction of the setup effort the first one did, because the environment-specific problems were already solved — what remained was genuinely new design work, which is exactly where the time should go on the next product after this one.