

DEVELOPMENT · 2026-10

Smart slice: a small search with clear limits

How Seesaw approaches a difficult resin model by trying bounded orientations, inspecting the result, and keeping the original project intact.



By Lyra

Published Oct 14, 2026 · 4 min read

October launch collection · Story context 2026-10-09



Automation feels helpful when it tries a sensible next step and explains the result. It gets uncomfortable when it quietly changes the problem until something passes.

Seesaw's October 9 Smart slice work takes a bounded approach to resin preparation. It tries a documented set of orientations with native supports, checks the actual sliced layers, and adopts a candidate only after the required validation. User choices such as scale, layer height, exposure, and material stay part of the input contract.

That makes this a useful developer story as well as a 3D-printing feature. A small, inspectable search can solve a concrete problem while leaving the person in control of the important constraints.

| Decide what the search may change

Start by listing the variables the algorithm owns. In this workflow, orientation and support preparation are candidates for change. The model's scale and printer/material settings are preserved.

Without that boundary, an optimizer can produce a misleading success. Reducing a model until it fits a limit may satisfy a technical check while violating the user's intended dimensions. Increasing layer height may shorten a job while changing the requested output.

Write preservation checks alongside the search. Compare the result's settings with the input snapshot before adoption. Treat an unexpected change as a failed attempt, even if the backend reports successful execution.

| Bound the attempt sequence

The [published Smart slice design](#) starts with the current orientation and tries a small sequence of relative tilts. Candidates outside the bed or the supported layer cap are skipped. Attempts are sequential and share an overall time budget.

Those limits make the feature easier to explain. The user can understand that it is exploring a specific preparation space rather than waiting for an undefined search to finish.

A general design sketch looks like this:

```
capture current input
for each allowed candidate within the budget:
    validate candidate bounds
    run native preparation
    inspect the actual output
    adopt only if inputs are still current and checks pass
preserve original project if no candidate is accepted
```

Backend failures and input-integrity problems should have explicit stop behavior. Retrying a broken executable through every candidate creates noise and wastes time.

| Read the successful fixture carefully

In the documented test, the unchanged challenge model failed without supports. Adding supports at the original orientation still left large islands. A five-degree X tilt with supports produced an accepted software result across 449 layers, including four strictly checked isolated-pixel removals.

That worked for this test model. Other geometry needs its own inspection, and the test's exposure settings still need calibration for the actual resin and printer.

The useful part is being able to follow the change: same model, a small tilt, new layers, and a checked result. The original input stayed available for comparison.

| Keep failure useful

A failed search should retain enough diagnostics to show which candidates were tried and why they were rejected. It should leave export blocked and preserve the original editable project.

Distinguish an inspectable intermediate slice from a validated printer export. Letting someone view a failed candidate can help debugging, provided the interface doesn't imply that the candidate is ready to print.

Cancellation and editing deserve their own tests. If the user changes the model while an attempt runs, the completed attempt belongs to the old input. Discard it rather than overwriting the user's newer work.

| Verify the surrounding workflow

The feature's value includes what happens after success: the selected preparation becomes visible, the user can inspect layers, the project can be saved, and Undo restores the prior state while clearing stale readiness.

Those interactions are easy to miss when testing only the search function. Exercise the actual button and the actual save/reopen path with a representative fixture.

A bounded search with clear preservation rules is a useful pattern for many tools. Define what may change, measure the candidate's real output, keep rejection reasons, and make the final adoption conditional on the current user intent.

| Sources & further reading

Checked 2026-10-14 · Story context 2026-10-09

1. [Seesaw Smart slice design and acceptance](#) (2026-10-14)

2. [Smart slice implementation commit](#) (2026-10-14)

[Read the current article online →](#)

Support the workshop

Explore software and learning offers at [StackSocial](#). Affiliate link: ProgreTech may earn a commission from purchases. Check current price, license and renewal terms before buying.

Find our projects at [progretech.com](#).

ProgreTech LLC · Build / Learn / Improve / Together

This file contains its text, illustrations and fonts. No account, scripts or network connection required to read. External reference links need internet access.