burin

User Manual

Version 1.0.efffae9 July 7, 2026

Contents

Introduction

burin is a Lisp-scripted solid modeler for designing 3D-printable parts. You describe a shape in a small Lisp script: spheres, boxes, cylinders, and the boolean operations that combine them. burin renders the result live in a preview window and exports it as a binary STL file, ready to slice and print.

A *burin* is an engraver's carving tool. The name fits: the core operation is carving one solid out of another.

How it's different

There's no drag-and-drop editor and no feature tree. A model is a text file.

A text file is easy to version, diff, and reuse. A parametric case design comes down to a handful of named variables, like wall thickness or screw spacing, referenced throughout the script. Change one number and every dependent surface updates.

It's also easy to drive from an AI coding assistant. burin exposes its scripting language over a small local HTTP API, so a coding agent such as Claude Code can write or edit a scene file and push it to the live preview by making HTTP requests. Section 8 covers how to set this up.

Core conventions

> **Note** **1.0 unit = 1.0 millimetre**, everywhere: in scripts, in the preview, and in the exported STL. Binary STL files carry no unit metadata, so this is the rule that gives the numbers in a script their meaning. `(sphere 10)` is a 10 mm-radius sphere and exports with a 20 mm cube bounding box.

> **Note** **Z is up**, the same convention CAD tools and slicers use. The XY plane at Z=0 is the "print bed" in the preview.

Launching burin

burin ships as two programs:

- **`burin`**: the windowed application, with a live raymarched 3D preview and the same HTTP API described in Section 7.
- **`burin_cli`**: a headless command-line tool for exporting STL files, running diagnostics, or running the HTTP API without a window. Useful on a server or over SSH.

The windowed application

```
./burin                                  # empty scene, HTTP API on :8080
./burin --scene scenes/sphere.lisp       # load a scene file at startup
./burin --scene scenes/sphere.lisp --port 9090
```

Headless (no window)

```
./burin_cli --serve 8080                  # HTTP API only, no window
./burin_cli --export scenes/sphere.lisp out.stl --cell 0.5 --decimate 0.01
./burin_cli --selftest                    # internal geometry self-checks
```

See Section 9 for the full flag reference.

The Preview Window

When a scene is showing, the window renders it live with a raymarched shader. Three reference grid planes cross the origin, with 10 mm minor lines and the X/Y/Z axes drawn in red, green, and blue, to help judge scale and placement. The model correctly occludes grid lines behind it.

Camera Controls

Input	Action
`Right-mouse drag`	Orbit around the target
`Middle-mouse drag`	Pan
`Scroll wheel`	Zoom

Side Panel

The side panel shows live statistics for the current scene: node count, tape row count, and rendering FPS. It also has an **Export STL** button. The button runs the same export code as the HTTP `/export` endpoint (Section 7), so the two produce identical files.

Writing Scenes in Lisp

A scene file is a plain-text Lisp script, evaluated top to bottom. Shapes are built from a small set of primitives, combined with boolean operations, and positioned with transforms. Whichever node you pass to `(show ...)` last is what the preview window renders and what gets exported.

Save scene files with a `.lisp` extension. The `scenes/` folder in the burin project is a convenient place to keep them. Treat the file as the permanent record of a design: the live preview window's state isn't saved anywhere else.

Primitives

Every primitive is centred on its own local origin. Move it into place afterwards with `translate`, `rotate`, or `scale`.

Form	Meaning
`(sphere r)`	radius `r`, in mm
`(box hx hy hz)`	**half**-extents in mm (a `(box 10 10 10)` is a 20 mm cube)
`(cylinder h r)`	`h` is the **half**-height, along the shape's local Z axis
`(capsule h r)`	`h` is the **half**-height, along the shape's local Z axis
`(torus R r)`	major radius `R`, tube radius `r`, ring lies in the local XY plane

Combining Shapes (Booleans)

Form	Meaning
`(union a b ...)`	merges any number of shapes into one
`(intersection a b ...)`	keeps only the space shared by every shape
`(difference a b ...)`	`a` minus (`b` union `c` union ...): carves the rest out of the first shape
`(smooth-union k a b)`	like `union`, but blends the seam with a fillet of radius `k` mm
`(smooth-difference k a b)`	like `difference`, with the same blended seam
`(smooth-intersection k a b)`	like `intersection`, with the same blended seam

> **Note** `union`, `intersection`, and `difference` take any number of arguments. The three `smooth-*` variants are binary: exactly two shapes plus the blend radius `k`. To blend more than two shapes, wrap smooth joins inside each other.

Placing Shapes (Transforms)

Form	Meaning
`(translate tx ty tz child)`	move by (`tx`, `ty`, `tz`) mm
`(rotate rx ry rz child)`	rotate `rx`/`ry`/`rz` degrees around X, then Y, then Z
`(scale s child)`	scale uniformly by `s` (non-uniform scale is not supported)

Scene Control

Form	Meaning
`(show x)`	set the live scene root to `x`: the only way a node becomes the thing that is rendered and exported
`(clear)`	empty the scene entirely and reset it, ready to build something new

The Underlying Lisp

Underneath the modeling forms above, scene files are ordinary Lisp. The usual small set of building blocks is available: `define`, `defun`, `lambda`, `let*`, `if`, `cond`, arithmetic (`+ - * /`), and comparisons (`< >`). Use them to make a script *parametric*: define a measurement once by name, then reuse it everywhere a dimension depends on it.

```
(define wall 2)                    ; every wall in this script is 2mm
(define outer-r 10)
(show (difference (sphere outer-r) (sphere (- outer-r wall))))
```

Basic Examples

A single box. Remember the arguments are half-extents, so this is a 20 mm cube:

```
(show (box 10 10 10))
```

A sphere.

```
(show (sphere 12))
```

A box with a corner bitten out of it, using `difference` to carve a second box out of the first:

```
(show
  (difference
    (box 10 10 10)                  ; the 20mm block we start with
    (translate 6 6 6 (box 6 6 6))))   ; carve a corner out of it
```

A hollow shell, combining a named variable with a `difference` of two concentric spheres:

```
(define wall 2)
(show (difference (sphere 10) (sphere (- 10 wall))))
```

A smooth (filleted) union of two spheres, so the seam between them blends instead of forming a hard crease:

```
(show (smooth-union 3 (sphere 8) (translate 10 0 0 (sphere 8))))
```

Modeling Tips

> Tip **Never let a cut sit exactly flush with the surface it pierces.** If you carve an open-top tray by subtracting a cavity box whose top face lands at exactly the same height as the outer box's top face, that seam is barely defined mathematically. It can show up as visible tearing or noise at the coincident boundary. Fix it by making the cavity, or any pilot hole or slot, overshoot the surface it cuts through: a few millimetres of margin for a large cavity, an extra millimetre for a small pilot hole.

> Tip **Check that parts meant to stay separate don't secretly overlap.** Two overlapping solids silently merge into one shape under `union`. The result is still a valid, watertight model, so nothing looks broken, but it isn't the two distinct parts you wanted. Before unioning shapes that should stay separate, compare the distance between their centres against the sum of their sizes.

Exporting to STL

An export can be triggered three ways, all running the same code path: the **Export STL** button in the preview window, the HTTP `/export` endpoint (Section 7), or the command line:

```
./burin_cli --export scenes/my_part.lisp out.stl --cell 0.3 --decimate 0.01
```

Export Parameters

Flag	Meaning
`--cell`	Grid size in mm used to sample the model (default `0.5`: a good balance of detail and speed). Use a finer value like `0.3` or `0.1` when a design has features smaller than about 5 mm, such as screw holes or thin walls, so they actually resolve. Coarser values (1–2) export faster but look faceted.
`--decimate`	Mesh-simplification tolerance in mm (default `0.01`; set to `0` to disable). Flat panels, like the sides of a boxy enclosure, would otherwise export as hundreds of needless coplanar triangles. Decimation collapses them close to the minimum triangle count needed, which keeps file sizes small with no visible change at normal print resolution.

> **Note** Exports are watertight, correctly wound manifold meshes and need no repair step before slicing. If you have `admesh` installed, run it over an export (`admesh out.stl`) as a sanity check: look for `0` disconnected edges and `0` backwards edges.

The HTTP API

Both `burin` and `burin_cli --serve` run a small local HTTP server, on port `8080` by default. The preview window's own Export button uses this server internally, and so can anything else that makes an HTTP request: a shell script, a build pipeline, or an LLM coding agent (Section 8).

POST /eval

Sends Lisp source to be evaluated against the live scene. The window (if one is open) updates immediately. Returns the printed result of the last form as plain text.

```
curl -X POST --data '(show (sphere 10))' http://localhost:8080/eval
```

A successful call prints the resulting node, for example `#<node 42>`. A response of `ERR`, or a failed connection, means the script has a problem. Check parentheses and argument counts against Section 4.

POST /export

Snapshots whatever the current scene root is and returns a binary STL, either streamed back in the response or written to a path on the server.

```
# stream the STL back:
curl -X POST 'http://localhost:8080/export?cell=0.5&decimate=0.01' \
     -o part.stl

# or write it directly on the machine running burin:
curl -X POST \
```

```
'http://localhost:8080/export?cell=0.5&decimate=0.01&path=/tmp/part.stl'
```

Using burin with an LLM Coding Agent

A burin model is a text file pushed over HTTP, so an AI coding assistant can design and iterate on parts for you. It edits a scene script, posts it to `/eval`, and you watch the result update live in the preview window. Export only once you're happy with the shape.

Agent Skills

Many LLM coding tools, Claude Code among them, support *skills*: a small Markdown file bundled with a project that tells the assistant when and how to use that project's tools. burin ships one at `skills/burin/SKILL.md`. It covers:

- where to save scene scripts (`scenes/*.lisp`)
- the workflow: write the file first, push it live, then export once you approve the result
- the modeling pitfalls from Section 5, so the assistant avoids them without being reminded each time

8.1.1 Claude Code

Claude Code auto-discovers a `skills/` folder at the root of the project it's running in. Nothing needs to be installed or configured. Open a Claude Code session with the burin project directory as the working directory, then ask for the part you want, for example "design a two-part enclosure for an ESP32-S3 board." The assistant picks up the skill on its own once it recognizes the request is about a burin part.

8.1.2 Other LLM tools

Any coding assistant that can run shell commands (to call `curl`) and read or edit files can drive burin the same way, even without native skill support:

1. Make sure `./burin` (or `./burin_cli --serve`) is running, so `/eval` and `/export` are reachable.
2. Give the assistant the contents of `skills/burin/SKILL.md` as part of its instructions or system prompt. This teaches it the DSL, the file layout, and the workflow above.
3. Ask it to design or modify a part in plain language. It should write or edit a `.lisp` file and push it with the same `curl` call shown in Section 7.

> **Tip** Describe changes in plain language and watch the preview window update. Ask for an STL export only once you're happy with what you see live.

Command Reference

burin (windowed application)

Flag	Meaning
`--scene <file.lisp>`	evaluate a scene script at startup
`--port <n>`	HTTP API port (default `8080`)
`--cam-yaw-deg <n>`	set the starting camera yaw, in degrees

Flag	Meaning
`--cam-pitch-deg <n>`	set the starting camera pitch, in degrees (clamped to ±89°)

burin_cli (headless)

Flag	Meaning
`--selftest`	run the internal geometry/export self-check suite
`--export <in.lisp> <out.stl>`	export a scene to STL; accepts `--cell` and `--decimate` (Section 6)
`--serve [port]`	run the HTTP API with no window (default port `8080`)
`--check-manifold <file> [cell]`	diagnostic: report marching-cubes mesh defects for a scene
`--check-tape <file>`	diagnostic: compare the CPU and GPU-preview evaluations of a scene

Copyright & Third-Party Notices

burin's Lisp interpreter is a fork of **TinyLisp** by Robert van Engelen (BSD 3-Clause License). The preview window is built on **raylib** by Ramon Santamaria (zlib/libpng license) and **Nuklear** by Micha Mettke (public domain / MIT dual license). Marching-cubes edge/triangle tables are from Paul Bourke's public-domain "Polygonising a scalar field," based on work by Cory Gene Bloyd. Full license texts for all of the above are included with the software in its `LICENSE` file.