

Concept for **dvi3v** project

v3.9.2.208

Peter Felecan

with Claude (Anthropic AI), model Sonnet 5

October 12, 2026

Contents

1	What is dvi3v?	2
1.1	Overview	2
1.2	Core Concept	2
1.3	Goals	3
2	Why dvi3v?	3
2.1	Problems Solved	3
2.1.1	Manual Tracing is Tedious	3
2.1.2	Dimensions are Critical	3
2.1.3	Photography Conditions Vary	3
2.1.4	Object Complexity	4
2.2	Use Cases	4
3	How Does It Work?	4
3.1	Workflow Overview	4
3.2	Key Technologies	5
3.2.1	Scale Detection	5
3.2.2	Object Detection	5
3.2.3	Morphological Processing	6
3.2.4	Vectorization	7
3.2.5	Dimension Calculation	7
4	Architecture & Design	8
4.1	Design Principles	8
4.2	Key Design Decisions	8
4.2.1	Decision 1: Bash + ImageMagick	8
4.2.2	Decision 2: Reference Paper for Scale	8
4.2.3	Decision 3: Center-Weighted Detection	8
4.2.4	Decision 4: Auto-Threshold	9

4.2.5	Decision 5: No-Holes Mode Optional	9
4.3	Error Handling Philosophy	9
5	Technical Specifications	10
5.1	Dependencies	10
5.2	Performance	10
5.3	Limitations	10
6	File Format	11
6.1	Output SVG Structure	11
6.2	Optional Exports (DXF, STL)	12
7	Project Artifacts	12
8	Theoretical Foundation	12
8.1	Image Processing Theory	12
8.2	References	13
8.3	Quality Metrics	13
9	Glossary	13

List of Figures

List of Tables

1 What is dvi3v?

1.1 Overview

dvi3v is a command-line tool that extracts objects from photographs and converts them into filled SVG (Scalable Vector Graphics) files with real-world dimensions in millimeters.

Input: Photograph of an object (JPEG, PNG, etc.)

Output: SVG file with accurate dimensions in mm

1.2 Core Concept

The tool bridges the gap between physical objects and digital vector graphics by:

1. **Detecting** the object in a photograph
2. **Extracting** its contour/silhouette

3. **Measuring** its real-world dimensions using a reference (paper sheet)
4. **Vectorizing** the shape into scalable SVG format
5. **Filling** with a solid color of choice

1.3 Goals

- **Accuracy without special equipment:** real-world millimetre dimensions from an ordinary photograph and a sheet of paper, not a scanner, a calibrated rig, or a depth sensor.
- **No manual tracing:** the object's outline is extracted automatically, not drawn by hand in vector-editing software.
- **Fabrication-ready output:** a filled SVG (and, optionally, DXF / STL) usable directly by CAD, laser-cutting, and 3D-printing workflows, not just a picture.
- **Works in ordinary conditions:** varying lighting, backgrounds, and object placement, on a single Bash + ImageMagick + **potrace** pipeline runnable from the command line, with no cloud service or GPU.
- **Known, documented limits:** where the tool falls short (thin features, the margin/small-holes trade-off, camera-angle assumptions), the limit is measured and stated, not silently wrong.

2 Why dvi3v?

2.1 Problems Solved

2.1.1 Manual Tracing is Tedious

Problem: Creating vector outlines of physical objects requires manual tracing in software like Inkscape or Illustrator.

Solution: Automated extraction from photos—snap a picture, get an SVG.

2.1.2 Dimensions are Critical

Problem: Simple image-to-SVG converters don't preserve real-world scale.

Solution: Uses reference paper (A4, Letter, etc.) to calculate precise mm dimensions.

2.1.3 Photography Conditions Vary

Problem: Real-world photos have varying lighting, backgrounds, and object positions.

Solution: Robust detection with auto-optimizing threshold, works with any background.

2.1.4 Object Complexity

Problem: Objects with holes (scissors, keys) or complex shapes are difficult.

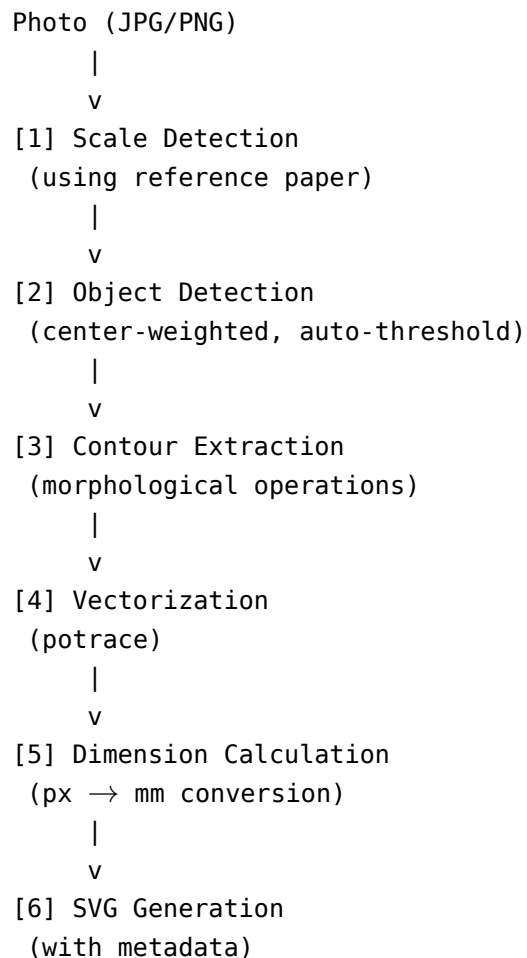
Solution: Optional hole-filling mode, morphological operations for cleanup.

2.2 Use Cases

- **CAD/Design:** Import physical object outlines into CAD software
- **Fabrication:** Create cutting templates for CNC, laser cutters
- **Documentation:** Archive object shapes with accurate dimensions
- **Prototyping:** Quick digitization of hand-drawn or physical templates
- **Education:** Teaching vector graphics, measurement, image processing

3 How Does It Work?

3.1 Workflow Overview



|
v
SVG File (real mm dimensions)

3.2 Key Technologies

3.2.1 Scale Detection

Challenge: Convert pixels to millimeters

Method: three detection strategies, tried in order until one validates against the expected paper aspect ratio:

1. **Automatic color detection** (default, when `--paper-color` is not given): the photo is reduced to its dominant colors by color clustering (ImageMagick's built-in quantization), and each large-enough color is tried as a match target, largest first.
2. **Named color match** (when `--paper-color` names a non-neutral color, e.g. `red`; also used internally by step 1 for each candidate): the photo is matched by color distance against that color's RGB value, with tolerance for lighting and camera white balance.
3. **Brightness match** (the only method for a neutral `--paper-color` — white, gray, light/dark gray, or black — and the fallback if 1–2 find nothing valid): the photo is converted to grayscale and searched for the largest sufficiently bright region of the right aspect ratio.

Once the paper region is found: measure its pixel dimensions, then calculate scale: `px_per_mm = pixels / known_mm`, applied to the whole image. If no valid paper region is found by any method, batch mode falls back to an EXIF-based estimate (subject distance and focal length, when available) or `--reference-mm` if given.

Formats Supported:

Format	Dimensions (mm)
A0-A6	ISO A-series
Letter	216×279
Legal	216×356
Tabloid	279×432

Paper Colors: any color name accepted by `--color` (see below) can be given via `--paper-color`; automatic detection (method 1 above) handles most colored paper without the flag at all.

3.2.2 Object Detection

Challenge: Identify the correct object among noise, artifacts, background

Method: candidate selection, then multi-factor scoring

At each threshold tried, two thresholding **polarities** are tried and the better-scoring candidate kept: "dark" (object darker than its background) and "light" (object lighter than

its background) — which applies cannot be assumed from the photo alone (a white object on colored paper needs "light"; most other cases need "dark"). Within each polarity, connected-component candidates are filtered by two safeguards before scoring:

- A candidate covering more than 70% of the frame is rejected outright — a legitimate, centered, not-cut-off object never fills most of the photo.
- A candidate whose bounding box contains more than 85% of the detected reference paper's own area is rejected — the paper is never legitimately "the object", even though it can be large and centered enough to otherwise look plausible.

The surviving candidate closest to center (with a bounded, non-runaway area term, so a bigger region is never preferred purely for being bigger) is picked for that threshold/polarity, then scored on 5 factors (0-100 points) for the **auto-threshold** comparison across thresholds:

Factor	Points	Purpose
Centrality	20	Prefer objects near image center
Size Ratio	20	Optimal size: 10-40% of image
Compactness	15	Avoid elongated artifacts
Fill Ratio	30	Measure object solidity vs fragmentation
Threshold Stability	15	Prevent extreme thresholds

Auto-Threshold:

- Tests N threshold values (default: 30-80% range)
- Evaluates each (both polarities) with the scoring system above
- Selects optimal balance between detection quality and stability

3.2.3 Morphological Processing

Purpose: Clean object edges, fill holes, add margins

Operations Used:

- **Close:** Fill small gaps, connect nearby regions
- **Open:** Remove noise, smooth edges
- **Dilate:** Expand object (for margins)
- **Erode:** Shrink object (part of Close/Open)

Hole Filling (`-no-holes` / `-n`, a boolean flag, no argument):

Method: Topological exterior flood-fill

1. Clean edges (morphology Close Disk:3)
2. Flood-fill the exterior background from a border corner, marking it; whatever the flood cannot reach is an enclosed hole

3. Turn every enclosed hole white (filled); restore the marked exterior to black
4. Despeckle (morphology Open Disk:2) to remove isolated noise pixels before the margin is added -- otherwise a single stray pixel becomes a disproportionate solid disk once dilated by the margin
5. Result: internal holes filled, of any size or shape, without eroding thin frames (unlike a fixed-radius morphological close)

Default (no -n): holes are preserved, not filled.

Usage: -n # fill all enclosed holes
(no strength/radius parameter)

3.2.4 Vectorization

Tool: potrace (bitmap to vector converter)

Settings:

- --tight: Minimize SVG file size
- --alphamax 0.8: Corner detection sensitivity
- --opttolerance 0.3: Curve optimization

Output: Single SVG path element with fill color

3.2.5 Dimension Calculation

Formula:

$$\text{dimension}_{\text{mm}} = \frac{\text{dimension}_{\text{px}}}{\text{px per mm}} \quad (1)$$

Reported at runtime (console output, and in `debug_info.txt` under -d; not embedded in the SVG itself — see *Output SVG Structure* below):

- Scale factor (px/mm) and detection method (paper-detected — and by which of the three paper-detection methods — EXIF-estimated, or user-provided via --reference-mm)
- Reference paper format and color (when paper-detected)
- Object dimensions (width × height in mm) and polarity (dark/light)
- Margin applied and fill color

4 Architecture & Design

4.1 Design Principles

1. **Simplicity:** Single bash script, no complex dependencies
2. **Robustness:** Works with realistic, uncontrolled photography
3. **Transparency:** Full error context, debug mode, metadata in output
4. **Flexibility:** Configurable paper, colors, margins, thresholds
5. **Automation:** Auto-threshold finds optimal settings

4.2 Key Design Decisions

4.2.1 Decision 1: Bash + ImageMagick

Rationale:

- Bash: Universal on Linux/Unix systems
- ImageMagick: Powerful image processing, already installed on most systems
- Core SVG extraction needs neither Python nor a compiled toolchain (Python 3 is used only by the optional `--stl` export's default backend; `--stl scad` uses OpenSCAD instead, and STL is skipped entirely, with a warning, if neither is available)

Trade-off: Less performance than compiled code, but sufficient for typical use

4.2.2 Decision 2: Reference Paper for Scale

Rationale:

- Common, available everywhere (A4, Letter)
- Known dimensions (210×297mm, 216×279mm)
- No special equipment needed

Alternative considered: QR codes with embedded scale → Rejected (too complex)

4.2.3 Decision 3: Center-Weighted Detection

Rationale:

- Users naturally place subject in center
- Reduces false positives (background objects)

Alternative considered: Machine learning detection → Rejected (overkill, dependencies)

4.2.4 Decision 4: Auto-Threshold

Rationale:

- Manual threshold selection tedious
- Optimal value varies per image
- Quality scoring enables automation

Implementation: Multi-factor scoring (v2.4+) with threshold stability

4.2.5 Decision 5: No-Holes Mode Optional

Rationale:

- Some objects need holes preserved (eyeglass frames)
- Some need holes filled (scissors handles, keys)
- User choice is best

Implementation: Flood fill from edges

4.3 Error Handling Philosophy

Principle: Fail clearly with context

Implementation:

- Global error trap with line numbers
- Function names in stack trace
- Command context on failure
- Automatic cleanup on error

Example:

```
=====
ERROR: Script failed with exit code 1
=====
Location: Line 432
Function: detect_object()
Command: convert "$INPUT_IMAGE" -threshold 50% ...
=====
```

This helps users debug and report issues effectively.

5 Technical Specifications

5.1 Dependencies

Required:

- bash 4.0+ (associative arrays)
- ImageMagick 6.9+ (`convert`, `identify`)
- potrace 1.15+
- bc (arbitrary precision calculator)

Optional:

- Standard Unix tools: `grep`, `awk`, `cut`, `sed`

5.2 Performance

Typical execution time:

Image Size	Time (seconds)
800×600 (small)	2-3
2000×1500 (med)	5-8
4000×3000 (lg)	15-25
Auto-threshold	+3-5

Bottlenecks:

- Morphological operations (Close, Open)
- Connected components analysis
- Potrace vectorization

5.3 Limitations

Technical:

- Single object detection (selects best one)
- Binary threshold (can't handle gradients)
- 2D only (no depth information)

Photographic:

- Requires minimum contrast

- Object should be near center
- Reference paper should be visible (for accurate scale)

Scale:

- Estimated scale if no reference (EXIF-based, $\pm 10\text{-}30\%$ error)
- User must provide `-r` or include reference paper

6 File Format

6.1 Output SVG Structure

```
<?xml version="1.0" standalone="no"?>
<!-- Generated by dvi3v v3.7.1.0 -->
<!-- Threshold: 65% (auto-optimized) -->
<!-- Mode: No holes (outer contour only) -->
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg version="1.0" xmlns="http://www.w3.org/2000/svg"
  width="46.35mm" height="94.22mm" viewBox="0 0 506.48 1029.51"
  preserveAspectRatio="xMidYMid meet">
<metadata>
Created by potrace 1.16, written by Peter Selinger 2001-2019
</metadata>
<g transform="translate(...) scale(0.1,-0.1)" fill="#000000" stroke="none">
<path d="M... z"/>
</g>
</svg>
```

The "Mode" comment line is present only with `--no-holes`. Scale, paper detection method, and object dimensions are reported on the console during the run (and in `debug_info.txt` under `-d`), not embedded as SVG comments.

Key attributes:

- `width, height`: Real dimensions in mm
- `viewBox`: Pixel dimensions for rendering
- Single `<path>` element: The object outline
- A few comment lines recording how it was generated (version, threshold, mode)

6.2 Optional Exports (DXF, STL)

The SVG is the primary output. Two optional companion formats can be written beside it (same basename):

DXF `--dxf` writes a millimetre-scaled DXF using potrace’s native DXF backend (no extra dependency), for CAD and CNC import.

STL `--stl` (and `--stl-thickness MM`) extrudes the silhouette to a flat plate of the given thickness. The default backend (`--stl python`) needs only Python 3: it uses the bundled `StlExtrude` helper, which triangulates with a vendored earcut and writes a watertight STL, so an end user does not have to install OpenSCAD. `--stl scad` uses OpenSCAD instead. If the chosen backend is unavailable the other is used as a fallback; if neither, the STL is skipped and the SVG is still produced.

7 Project Artifacts

The tool itself (documented above) is the core of the project, but not the whole of it. Each of the goals above is backed by other work, documented separately:

- **Accuracy** is measured, not assumed: [Test Plan](#) lays out the validation campaigns (materials, shapes, sizes, lighting, real-world objects) that quantify how close the extracted dimensions come to the physical ones, and [Reference Calibration Card](#) specifies (and, for the ISO chart variant currently in use, provides) the known-dimension targets those campaigns measure against.
- **Working in ordinary conditions** still benefits from a **consistent** ordinary condition: [Camera Document Stand](#) documents the physical jig used to hold camera, object, and reference paper in a repeatable arrangement for testing, and [Manufacturing Configuration](#) records the exact settings used to print it, so it can be reproduced.
- **Fabrication-ready output** and **known limits** are both claims that need checking against what else exists: [State of the Art](#) surveys comparable tools and shows where `dvi3v`’s combination of reference-derived scale, automatic extraction, and vector output is (and is not) unique.
- Two process documents support the above without being goals themselves: [Data Exchange Workflow](#) (how test photographs and results move between test execution and integration without re-sending the whole project) and [Debug File Upload](#) (what to capture and share when something needs diagnosing).

8 Theoretical Foundation

8.1 Image Processing Theory

Threshold Selection:

- Based on Otsu’s method (optimal threshold for bimodal histograms) [Otsu 1979](#)
- Extended with quality scoring for non-ideal conditions

Morphological Operations:

- Mathematical morphology [Serra 1982](#)
- Structuring elements: Disk (circular, isotropic)

Vectorization:

- Potrace algorithm [Selinger 2003](#)
- Bézier curve fitting to bitmap edges

8.2 References

- N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. [doi:10.1109/TSMC.1979.4310076](https://doi.org/10.1109/TSMC.1979.4310076)
- J. Serra, *Image Analysis and Mathematical Morphology*. London: Academic Press, 1982. ISBN 0-12-637240-3.
- P. Selinger, "PTRACE: a polygon-based tracing algorithm," technical report, September 20, 2003. potrace.sourceforge.net/potrace.pdf

8.3 Quality Metrics

Fill Ratio: Measures object compactness

$$\text{fill ratio} = \frac{\text{white pixels}}{\text{bounding box area}} \quad (2)$$

Threshold Stability: Gaussian-like preference curve

$$\text{stability}(t) = \text{max points} \times e^{-\frac{(t-\text{optimal})^2}{2\sigma^2}} \quad (3)$$

Approximated as piecewise linear in implementation.

9 Glossary

Term	Definition
Continued on next page	

Continued from previous page

Term	Definition
Binary image	Black and white image (1-bit per pixel)
Connected components	Contiguous regions of white pixels
Morphological operation	Shape-based image transformation
Potrace	Bitmap to vector conversion algorithm
SVG	Scalable Vector Graphics (XML-based vector format)
Threshold	Cutoff value to convert grayscale to binary
Vectorization	Converting bitmap to vector paths

The physical camera document stand used to capture the project’s test photographs (design, manufacturing, and dvi3v-workflow integration) is documented separately: see [Camera Document Stand](#).