Trace2SCAD:
Converting images into parametric OpenSCAD models

Henry Gordon Dietz; Department of Electrical and Computer Engineering, University of Kentucky; Lexington, Kentucky

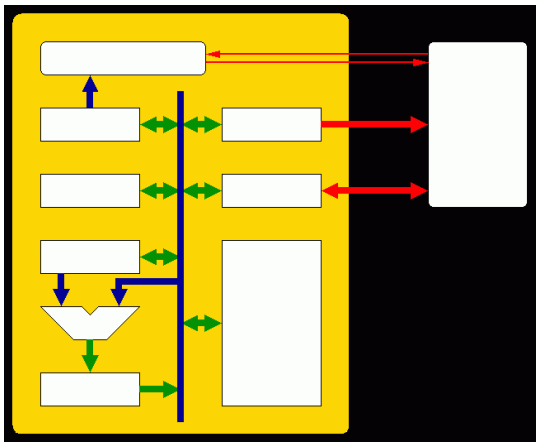

Figure 1. *A simple computer architecture diagram*

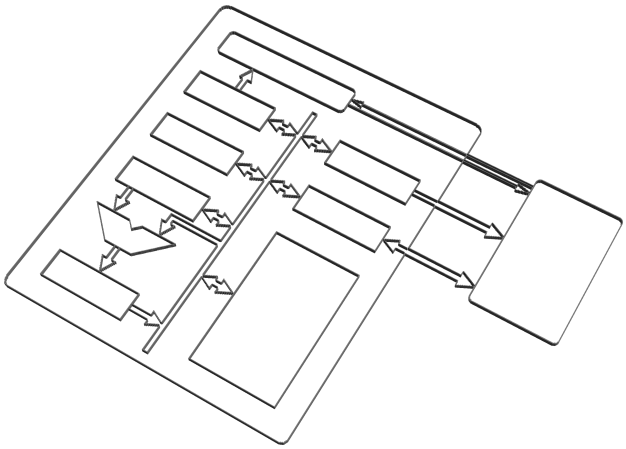

Figure 2. *Figure 1 rendered for a blind student by Trace2SCAD*

Abstract

Ironically, it is often useful to manipulate 2D graphics for 3D printing. Trace2SCAD was originally written to convert 2D diagrams into printed structures that could be understood by a blind student feeling the surface. However, by its initial open source release in January 2015, it had already been applied to convert pixel-mapped images into parametric 3D models and 3D-printable image renderings (e.g., lithophanes). The original version was a shell script using a variety of helpers, especially potrace. The new version is pure C++ code using the OpenCV library and incorporating many new features, including the ability to render color images as 3D models to be printed for viewing by reflected light (similar to the filament paintings produced by HueForge). It is also significant that the output generated is not mere 3D shapes, but parametric 3D models coded in the OpenSCAD programming language. This paper describes the functionality and algorithms used in the new Trace2SCAD.

Introduction

Additive manufacturing (AM) is the concept of constructing physical objects by building-up material. There are a wide range of AM technologies, from stacking cut sheets of paper to using a laser to selectively melt metal powder. However, the technique commonly known as either fused deposition modeling (FDM) or fused filament fabrication (FFF), extruding a heat-softened plastic, quickly became the dominant technology and remains so today. In a typical FDM 3D printer, a moving printhead extrudes plastic filament through a hot nozzle, depositing plastic to cool and harden at the X, Y, Z coordinates commanded. The process of converting a 3D model into a series of movements in the X, Y, Z, and E (extrusion material feed) dimensions is called slicing because the 3D model is usually sliced into layers so that only the X, Y, and E dimensions are changing as each line of filament is extruded. Layers of material are built up in the Z dimension upon a flat print bed, with each layer supporting the next.

The dominant plastic extruded was for a time acrylonitrile butadiene styrene (ABS), a fairly opaque material best known to many as the stuff Legos are made of. Although this material is easily heated and extruded, it produces reasonably durable parts that do not soften until around 80°C. However, in FDM printers, ABS parts tend to warp due to shrinkage during cooling and the fumes from hot ABS and airborne ultrafine particles (UFPs) of non-biodegradable plastic are potential health concerns. As a result, a compostable translucent bio-plastic made from renewable materials, polylactic acid (PLA), has taken over as the dominant FDM filament. PLA warps far less than ABS, and the fumes at printing temperatures are less of a concern, but objects made of PLA soften and can deform at temperatures as low as 50°C. It was largely the low softening temperatures of ABS and PLA parts that made 3D printing be viewed primarily as a technology for rapid prototyping rather than making mechanical parts for long-term use.

However, as 3D printers became commodity items over the last decade, people found many new ways to use them. This broader range of applications is bolstered by the availability of an increasingly wide range of extrudable plastics with various characteristics. There is an old saying that observes: "If your only tool is a hammer then every problem looks like a nail." It is

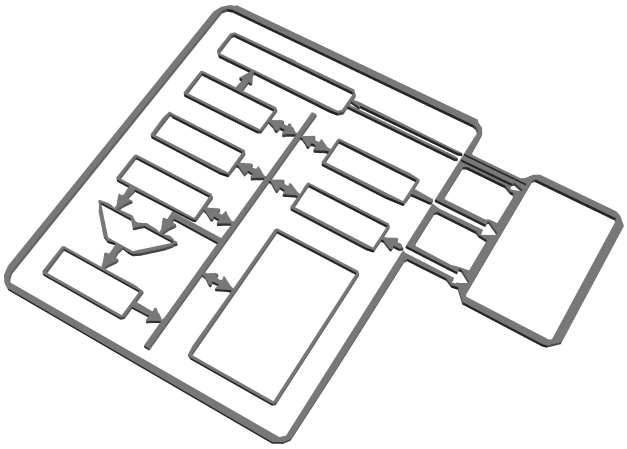

Figure 3. *Figure 1 rendered with fewer than 512 points*

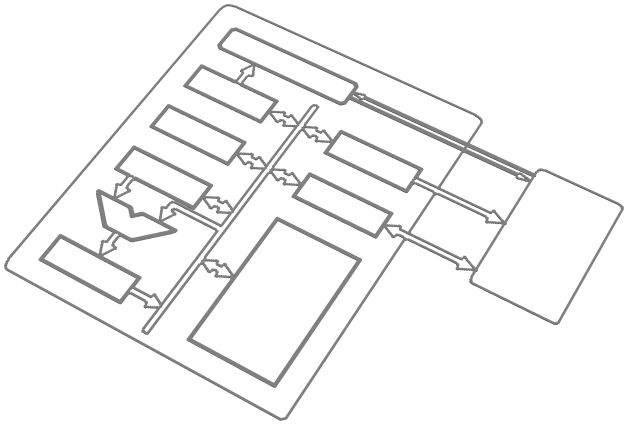

Figure 4. *Figure 1 rendered by the new Trace2SCAD*

easy to have a 3D printer in your business or home, so FDM has become the hammer nearly every "maker" has and FDM is being used to make a lot of things that aren't rapid prototypes nor even fundamentally mechanical parts.

This is where tools like Trace2SCAD come in. Trace2SCAD is intended to be a nearly universal preprocessor to convert 2D images into 3D models suitable not just for making mechanical parts, but also for rendering 2D images in novel ways. Note that Trace2SCAD is not using generative artificial intelligence methods in the transformations it performs, but is simply rendering with the goal of enabling an FDM 3D printer to duplicate attributes of the 2D image as accurately as possible. The recent proliferation of filament changers that allow an FDM printer to automatically swap between different filaments means that FDM is now easily capable of printing painting-like images using multiple colors of filament.

Before discussing how Trace2SCAD works, it is useful to explain a key property that is reflected in the name: Trace2SCAD is not a an image-driven slicer, but a generator of parametric 3D models expressed in the OpenSCAD programming language. In truth, the 3D models output by Trace2SCAD are not really 3D, but 2.5D structures in which the Z dimension is treated as a stack of distinct layers – which sounds very much like slicing. However, expressing those layered structures as parametric 3D models allows significant further transformations to be performed. Trace2SCAD also incorporates a variety of methods aimed at reducing the 3D model complexity to facilitate additional processing.

The following sections discuss the ideas and algorithms used by Trace2SCAD for different types of tasks. The first discusses the task for which Trace2SCAD was originally built: the extraction of a vectorized 2.5D model from a pixel-based image, an example of which is shown in Figures 1-4. Lithophanes use different thicknesses of extruded plastic to replicate an image when backlit. The next section discusses creation of monochromatic lithophanes and the section after that discusses color lithophanes and filament painting. The final section summarizes the contributions.

Vectorization

Most digital images are pixel maps, and most image-processing software tools operate on pixel maps. A pixel map generally is a rectangular array of pixels in which the value of each pixel represents a spatial region's appearance. There is some ambiguity about the shape of the spatial region described by a pixel. Most software treats pixels as dimensionless points, but the point value represents the average value over some area, and the shape of that area is sometimes treated as a square and sometimes treated as a more complex point spread function (PSF), perhaps with a diffuse Gaussian distribution. In any case, the spatial resolution of a pixel map is fixed, and not scalable without introducing assumptions – i.e., artifacts.

Vectorization is the process of identifying edges within a pixel map and grouping adjacent similar-value pixels into regions outlined by a series of line segments (vectors) that are trivially scalable to any desired size. Some vectorizers take this concept one step farther and replace the vectors with curves to produce a smoother boundary. However, the pixel shape issue remains in the form of the presumed thickness of a line. For example, suppose a region exists that is the set of pixels at coordinates (22,42), (23,42), (24,42). It is quite easy, and obviously incorrect, to create a zero-width region for these pixels because the vectors from (22,42) to (24,42) and from (24,42) to (22,42) precisely overlay each other. Interestingly, this incorrectness is not obvious using a 2D drawing system because such systems will normally draw each line with a thickness, whereas the slicers for 3D printers try to fill-in between the lines.

Vectorization in the 2015 Trace2SCAD

Vectorization is commonly used in 2D editing as implemented by software like InkScape[1], so it is useful to examine how they handle the problem. The most common answer is that they run Potrace[2] as a helper application, so that is also what the original Trace2SCAD does.

The algorithm used in Potrace[3] is quite sophisticated, breaking the vectorization problem into four main stages. In the first stage, edge paths are defined. The second stage then converts these paths into polygons, and Potrace uses a clever scheme to reduce the problem of finding optimal (minimum vector com-

ponent count) vector outlines into finding an optimal cycle in a directed graph, which it then solves in $O(nm)$ time where n is the input path length and m is the length of the longest path. This non-local optimization of the vectorization is a key benefit of Potrace, but it also means that the vectors outlining two adjacent regions do not always align, resulting in small gaps and overlaps at region interfaces.

The last two phases of Potrace involve converting the polygons to smooth Bezier curves and optionally merging Bezier curves when their curvature allows. Although OpenSCAD's programming model allows writing Bezier curve functions (and libraries implement this[4]), it does not directly support objects defined by Bezier curves and thus the Bezier computations significantly slow OpenSCAD manipulation of a 3D model. Perhaps more significantly, neither most slicers nor the STL 3D model files that are typically fed into them support Bezier curves. In fact, STL files use triangular patches to define the surfaces of all objects. Thus, it seemed most appropriate for Trace2SCAD to directly use polygons.

The shell script that implements the original Trace2SCAD does some preprocessing of the image using ImageMagick[5], and then uses Potrace to obtain the polygonal region outlines. The order of polygons is significant because the goal is defining regions, not simply outputting vectors. Thus, Trace2SCAD includes a Awk script that orders the polygons so that nested polygonal regions can be subtracted from the polygonal regions that contain them, which the output OpenSCAD does using the `difference()` operator. However, Potrace's vector optimization can generate huge numbers of vectors for high-resolution images, so the Awk script also attempts to simplify any vectorization that exceeds a user-specified complexity target. This is done in two steps:

1. Simplify the polygons by replacing adjacent vectors that have similar angles with a single vector.
2. If the target complexity goal is not reached, rescale the image to a smaller size and repeat the process.

For example, to create the 3D model shown in Figure 2, Trace2SCAD first modified the image shown in Figure 1 by applying a default high-pass filter to identify edges. This is the default, rather than using the image directly, because the edges are what is needed for a blind person to understand the diagram by touching a 3D print. However, the filtered image edges are not lines; they are regions that have an obvious nested structure. The polygon output from Potrace had 4914 points, but Trace2SCAD was able to simplify that to just 2361. In a second run, Trace2SCAD was given a command line option to target complexity not exceeding 512. After under four seconds of user time and twenty passes of automatically rescaling the image, Trace2SCAD produced the model shown in Figure 4, which initially had 1034 points and was reduced to 510 in Trace2SCAD.

Vectorization in the new Trace2SCAD

The new Trace2SCAD does not use Potrace and ImageMagick, or any other helpers. Instead, it is pure C++ code using the OpenCV library[6]. The algorithm used to walk region edges to create a polygon is very straightforward, but fundamentally differs from that in Potrace in that it gives pixels unit width rather than treating them as points. This treatment avoids creating zero-width regions, which are more problematic for 3D printing than one might expect. The main problem with zero-width regions lies in the algorithm used by slicers to distinguish inside from outside given an STL file that only defines triangular surface patches. Position (X,Y,Z) is inside an object *iff* a line going from any point outside the bounding box of the STL must pass through an odd number of surfaces, but coincident surfaces may be incorrectly counted as a single intersection, thus confusing inside and outside. In 3D printing terminology, this is one of several ways in which a design can be non-manifold, and hence cause slicing to fail.

Figure 5. *The EI26 logo and vectorized 8-level SCAD models*

The new Trace2SCAD also does not search for the global minimum number of vectors to represent each polygon, but simply combines adjacent vectors that have similar angles. The allowable error in merging vectors can be controlled, and a setting of zero error essentially ensures that polygons for adjacent regions will not have gaps or overlaps. This unfortunately implies that the number of vectors can be greater and regions may have a rougher surface than with the Potrace-based vectorization, but the new Trace2SCAD reduces model complexity (although not necessarily model file size) by smarter handling of nested regions.

OpenSCAD coding of vectorized images

To explain the difference between the original and new Trace2SCAD vector codings, consider the EI26 logo shown in Figure 5. Both the original and new versions of Trace2SCAD are able to produce vector renderings of this image as OpenSCAD code, and these are also shown in Figure 5. For this example, high-pass filtering is disabled and the command line option `-l 8` has been used to specify creating vectorizations from the image using 8 equally-spaced gray levels to define regions. In both cases, the resulting model contains OpenSCAD `module` definitions for each gray level, although there are minor differences in the gray level determination due to a more sophisticated model of color being employed in the new version. For example, the 3919-point `module` defined for layer 6 using the original Trace2SCAD can be summarized as:

```
module layer6() {
 ...
 scale([1/maxd, 1/maxd, 1])
 translate([-minx-dx/2, -miny-dy/2, 0])
 linear_extrude(height=1, convexity=3919)
 union() {
  union() {
   union() {
    difference() { polygon(...); ... }
    polygon(...); ...
   }
   polygon(...); ...
   difference() { polygon(...); ... }
   polygon(...); ...
   difference() { polygon(...); ... }
   polygon(...); ...
   difference() { polygon(...); ... }
   polygon(...); ...
   union() {
    difference() { polygon(...); ... }
    polygon(...); ...
   }
   polygon(...); ...
}}}
```

The `union()` operations represent multiple disjoint regions that have the same gray level being combined. Nested regions are subtracted from those that enclose them using `difference()` operations. In contrast, the new Trace2SCAD defines a separate `module` for each region. The definition of a region contains the `polygon()` describing the region edge, but all the enclosed regions are differenced from it by invoking their modules.

Aside from 3D printing graphics, vector models are extremely useful for constructing mechanical parts to be compatible with complex structures. For example, there are many different camera mounts for lenses, quite a few of which include bayonet wings, lock notches, contact positions, and other features that must be precisely shaped and positioned. One of the easiest ways to make a model starts with taking one or more photos of the desired structure. It is best for these photos to be taken with a long lens that produces negligible distortion and to carefully align the camera with the part using a copy stand. Each image then can be edited to clearly distinguish the different surfaces, processed by Trace2SCAD to produce OpenSCAD models of the surfaces, and then scaled to match an easily accessible measured dimension of the part.

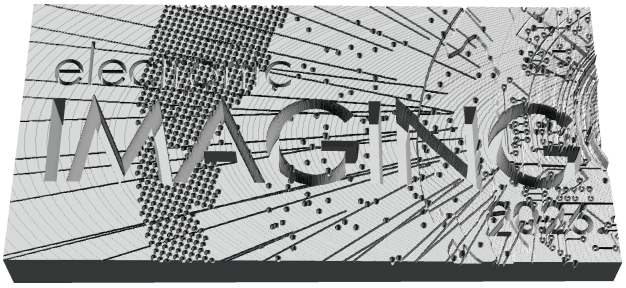

Figure 6. *The EI26 logo rendered by OpenSCAD `surface()`*

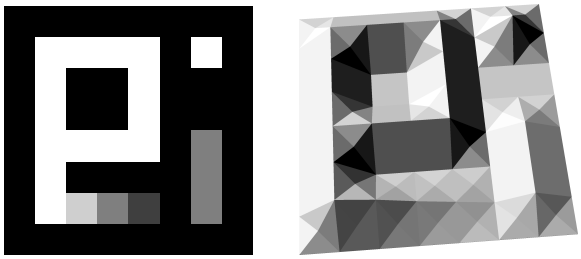

Figure 7. *An 8x8 image rendered by OpenSCAD `surface()`*

Monochrome lithophanes

A lithophane is a 3D object in which the thickness of the material is modulated so that viewing the backlit object will reveal a monochromatically shaded image. The use of backlighting implies that the filaments used must have some degree of translucence; the more opaque the filament, the thinner the object must be to let any backlight pass. It is also generally assumed that the FDM-printed filament will use 100% infill, so that the backlit brightness is only a function of thickness. Although lithophanes are usually modeled as having a flat side sitting on the printer bed, most printers can print finer and smoother thickness variations in the X and Y dimensions than in Z, so it can be advantageous to print the lithophane standing on a edge or, to avoid unsupported overhangs, tilted at 45° with removable supports underneath. With this description, it should be clear that the multi-level vector models in Figure 5 not only can be treated as 2.5D mechanical models or carved surfaces, but are also straightforwardly printable as lithophanes.

However, there are alternative ways to render a 3D model for a lithophane. Many slicers, and OpenSCAD itself, can read a 2D image to produce a bump map that can serve as a lithophane. For example, the OpenSCAD `surface()` operation was used to create the monochromatic lithophane model shown in Figure 6. Most slicers and OpenSCAD produce a bump map by treating each pixel's gray level as a height, typically from 0 to 255, so the model shown here was produced by first inverting the colors of the EI26 logo. Rather than rendering each pixel as a rectangular volume or creating optimized polygon regions, each pixel's value is used as the height of a point in the center of the pixel and

triangular surface patches are built to connect to the neighboring points. Figure 7 shows an 8x8 pixel image and the 3D model produced by OpenSCAD `surface()` (scaled to reduce the spike heights for this figure). The resulting model thus has complexity directly proportional to the number of input pixels, but more significantly contains slopes wherever the gray-level of a pixel differs from its neighbors. This is highly undesirable for making precisely-shaped mechanical parts, but tends to produce continuous, yet sharp, 3D features that can be treated as carvings or lithophanes. It is noteworthy that this is essentially a low-pass filtering of the pixel data, so thin details are not rendered with the correct average thickness.

Transmission distance

Of course, the underlying assumption above is that the tonal progression from 0 to 255 in an image is well approximated by a linear mapping to thickness. The basic visual linearity of tone mapping is commonly controlled by a parameter called gamma in image editing, so it makes sense that rendering to a lithophane should also allow gamma adjustment. Beyond that, filament opacity plays a key role in determining the tone produced by a given thickness of filament. The HueForge filament painting software[7] needs to know the apparent color of a stack of filaments that are not fully opaque, and for that purpose uses, and has popularized, an opacity metric called TD (transmission distance). Intuitively, the TD of a filament is the maximum thickness, in mm, of that material through which approximately 10% of the light will pass. There are TD measurement devices that essentially standardize this metric[8]. The device cited also determines the RGB color of the filament.

It is not entirely clear how the tonality of a given thickness of filament should be calculated from the TD and RGB color. Beer-Lambert essentially observed that absorbances of a stack of materials add while the transmittances multiply. However, this doesn't produce a precise formula for computing color (tone) as a function of TD and thickness. It is also worth noting that the backlit and frontlit color blending computations are inherently different; backlighting passes through the layers once, whereas front lighting must travel to each non-surface layer and back. Other complications include the fact that a material can have a different TD for each basic color component and RGB is not a colorspace where changes in different color component values make perceptually similar changes as seen by a human observer. For the backlit layers of a monochrome lithophane, relatively simple analysis should suffice with transmitted brightness decaying roughly exponentially as a function of thickness – and gamma values allow tuning this.

Voxels and palettes

Building on the TD concept, the new version of Trace2SCAD provides an alternative method to produce monochrome lithophane models. Instead of creating polygonal regions, Trace2SCAD can produce optimized voxel models with a user-determined voxel size. In other words, the voxel analysis is done at printer resolution given the desired lithophane dimensions and print layer height. This uses an entirely different code path within Trace2SCAD from the polygonal region analysis and the generated OpenSCAD models have a very different structure that again leverages the fact that OpenSCAD is a programming language. A palette is generated that specifies the color created by each allowable stack depth using the specified printer layer height and filament TD. Then a map is generated which specifies which palette entry, and hence which stack height, to use for each X,Y position's voxel. The map is created based on minimizing tonal errors. This analysis is very much like that used in computing optimal halftones[9, 10], and can even incorporate error diffusion so that fractional-layer tonal errors are statistically minimized over larger regions.

Color lithophanes and filament painting

The concept of a color lithophane would seem to be trivially modifying the concept of a monochrome lithophane, but stacks of translucent extruded layers do not just darken the backlight: they also can blend colors. With sufficiently translucent filaments, this color blending can also be seen in prints viewed by front lighting. The process of creating a color 3D print designed to be viewed with front lighting is called filament painting, and the process of creating a filament painting differs from making a color lithophane mostly in the math used to compute the combination of resulting blended colors. Neither realistically-colored lithophanes nor filament painting were very popular until 2023.

Although FDM 3D printers capable of automatically switching between filaments existed before the 2022 release of the Bambu Lab X1-Carbon (X1C) with its AMS (automatic material system), that system qualitatively changed how accessible multicolor 3D printing was at the consumer level. The Bambu AMS could hold 4 spools of filament, and up to 4 AMS units could be connected to one X1C printer, so automatic switching between up to 16 colors was viable. There are issues with Bambu's AMS taking too much time and wasting filament with priming every time a color is changed, but an X1C with an AMS only cost $1,449 and printed quickly and reliably without user tuning. Within a year, nearly every FDM printer maker announced a competitive filament changer.

The software support for design of realistically-colored lithophanes or filament painting was also missing key components until the release of HueForge[7] in 2023. The main contributions of HueForge are the TD metric and a graphical user interface that allows easy manipulation and visualization of filament assignments to depth ranges. The combination of HueForge and an AMS made color lithophanes and filament painting a viral success. In 2026, it is still a hot topic as hot-end changers make filament swaps cheaper; just as EI26 was being held, it was announced that there is now a version of the Orca slicer that supports "virtual filaments"[11] using this same blending concept.

In effect, the HueForge model is a layered decomposition of the types of monochrome lithophanes built by tools like the OpenSCAD `surface()` operator. HueForge allows the user to build and maintain a database of available filaments with TD and RGB color for each. Using selected filaments, it then allows interactive graphical control over precisely what height ranges in the print will use each filament. It is a flexible and powerful interface, but coloring is still a manual operation with minimal automation. The output is a bump map 3D model accompanied

Figure 8. *GA-optimized 4-filament, 10-layer pure stack filament painting*

Figure 9. *GA-optimized 4-filament, 8-layer voxel filament painting*

by a list of layer heights at which to change filament.

In contrast, there are no interactive controls in the new Trace2SCAD. It instead attempts to fully automate the rendering process using genetic algorithms to evolve high-quality filament-stacking solutions. There are two types of filament stacking, both supported and fully automated for either color lithophanes or filament painting:

1. True stacking (i.e., as implemented in HueForge) such that each layer of the 3D print contains only one color of filament
2. Arbitrary volumetric stacking with a maximum of s filaments used in the stack, but any of those s filaments may appear in disjoint portions of the same layer; this allows better color approximations for 3D printers that have filament changers with a capacity of s filaments

Trace2SCAD accepts a database of filaments with color and TD data. However, it automates the filament selection and stacking processes, and works primarily with colors as 32-bit values. A steady-state Genetic Algorithm (GA)[12] controls the search for the best palette. Each color in a palette corresponds to a stack of filaments. Each potential palette is evaluated for fitness by:

- The blended color of a stack of filaments is computed much like in HueForge, but using a somewhat different approximate formula; the best formula choice is still an open question
- Either K-means[13] or a second (nested) GA optimizes the color palette selection using JND (just noticeable difference) filtering to reject any stack that is too similar to a less costly stack
- Simple error propagation[14] can optionally be applied to improve average area color
- The image rendered using that palette is compared to the desired image appearance using the CIE DeltaE 2000 metric[15]

If a new most-fit palette is found, it is noted, and the search is ended if the quality is high enough. Otherwise, tournament selection is used to select good and bad population members. The worst of the tournament is then replaced with either a crossover product or mutation of the best from the tournament, and the search continues.

One would expect that the resulting palettes generally would be better for arbitrary volumetric stacks than for pure single-color-per-layer stacks, and empirically that is the case. For example, Figure 8 shows the the pure stack rendering of the EI26 logo from Figure 5 as both the theoretical image result and an actual 3D print. This rendering uses 4 filament colors and 10 layers, which means it had a palette with just 10 colors. Compare the Figure 8 rendering to that in Figure 9, which results from general volumetric stacks. Despite using the same 4 colors as the pure stack and only 8 instead of 10 layers, it has a palette of 28 colors and the rendering is significantly better. Unfortunately, the additional filament changes nearly double print time on the X1C.

From Figures 8 and 9 it is clear that the accuracy of the color blending model is acceptable, but there is plenty of room for improvement. The 3D prints photographed here have some stringing visible which shows primarily as thin red or white lines, but this is a common 3D printing artifact that was caused largely by the red and white filaments being quite old. Stringing is not a flaw in the model and these artifacts can be removed with relatively simple postprocessing of the print.

Comparing these prints to the original EI26 logo in Figure 5, it is clear that the four filaments used do not provide a way to duplicate the cyan background color of the EI26 logo, so both palettes used blue instead. These prints were made 208mm by 104mm with 0.2mm layers, and that relatively thick layer height is why the tonal gradations are less subtle than in the original image. In general, filaments with very high TDs printed at layer heights around 0.08mm are recommended to obtain smoother blending. Subtractive primary (yellow, magenta, and cyan) filaments can most accurately paint arbitrary colors.

Conclusion

This paper has discussed the basic philosophy and approach used in the original and new open-source Trace2SCAD software tools. The original was a shell script with various helpers; the new version is self-contained C++ code that supports a much wider range of 2D-image-to-3D-model conversions. Over 20 versions have been built and evaluated, and the process continues.

Given that it was 2025, we began not by writing new code for Trace2SCAD, but by attempting to use various AI systems to implement improved algorithms. We found that the AI assistants excelled at identifying relevant existing algorithms and were helpful in coding them, especially when implementation was dominated by reformatting data to allow calling an existing library routine. However, they did not produce any useful new insights and virtually the entire application is hand coded.

The main contributions here are a new algorithm for vectorization that treats pixels as unit areas and a family of new voxel-based filament color blending algorithms using CIE DeltaE 2000 color distance computations, K-means and a Genetic Algorithm for optimizing colors of lithophanes and reflected-light viewed filament paintings, and a variety of new optimization methods that reduce complexity of the OpenSCAD model generated (largely taking advantage of the fact the OpenSCAD is a programming language, so the 3D model generated is a parametric program rather than an STL).

Trace2SCAD is freely distributed from:

`https://aggregate.org/MAKE/Trace2SCAD`

References

[1] url: `https://inkscape.org`

[2] Peter Selinger, "Potrace: transforming bitmaps into vector graphics", url: `https://potrace.sourceforge.net/`

[3] Peter Selinger, "Potrace: a polygon-based tracing algorithm" 2003, url: `https://potrace.sourceforge.net/potrace.pdf`

[4] "LibFile: beziers.scad" url: `https://github.com/BelfrySCAD/BOSL2/wiki/beziers.scad`

[5] Michael Still, *The definitive guide to ImageMagick*, Berkeley, CA: Apress, ISBN 1590595904, 2006

[6] G. Bradski, "The OpenCV Library," Dr. Dobb's Journal of Software Tools, 2000

[7] Steve Lavedas, url: `https://shop.thehueforge.com/blogs/news/what-is-hueforge`

[8] url: `https://ajax-3d.com/`

[9] Robert Ulichney, *Digital halftoning*, MIT press, 1987

[10] Daniel L. Lau and Gonzalo R. Arce, *Modern digital halftoning*, CRC press, 2018

[11] Matthew Mensley, "Forget 4 Colors: This Orca Slicer Fork Unlocks Virtually Unlimited Color Combos" March 5, 2026, url: `https://all3dp.com/4/new-orca-slicer-fork-unlocks-virtual-color-printing-for-any-3d-printer/`

[12] A. Lambora, K. Gupta and K. Chopra, "Genetic Algorithm - A Literature Review" 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 2019, pp. 380-384, doi: 10.1109/COMITCon.2019.8862255

[13] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam, "The k-means algorithm: A comprehensive survey and performance evaluation" Electronics 9.8 (2020): 1295, doi: 10.3390/electronics9081295

[14] R. W. Floyd and L. Steinberg, "An adaptive algorithm for spatial grey scale" Proc. Soc. Inf. Display, 17:75-77, 1976

[15] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal, "The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations" Color Research & Application 30.1 (2005): 21-30