EGR 115 Final Project Report – MATLAB SVG Color Transformation Tool

Daniel E. Janusch

December 8, 2024

1 Background and the SVG Format

SVG files, or Scalable Vector Graphics files, are an image format unlike traditional "raster" image formats like PNG, JPEG, or BMP. While raster formats represent images as a grid of pixels, vector formats including SVG, PDF, and EPS, use mathematical descriptions like lines, curves, and shapes, making them resolution independent. There is a notable gap of good tools for working with SVGs programmatically, and many existing solutions have arbitrary limitations or don't preserve the advantages of vector graphics. For instance, ImageMagick [1] can work with SVGs, but it just rasterizes them first, which defeats the entire purpose of vectorization in the first place.

The SVG 1.1 format is governed by a web standard [2], and follows an XML format. The main ways that color can be given in an SVG is through `stroke` colors, `fill` colors, and embedded `<image>` elements. Embedded images can be PNG, JPEG, or WEBP, and can be either included inline or referenced through a file, however this project focuses on inlined images for simplicity. `fill` and `stroke` colors can be any of the following formats: #RGB, #RGBA, #RrGgBb, #RrGgBbAa, rgb(R,G,B), rgb(R%,G%,B%), rgba(R,G,B,A), rgba(R%,G%,B%,A), hsl(H,S%,L%), hsla(H,S%,L%,A), any valid named color [3], "none", or "transparent". The alpha values in the function forms can always be either a regular number or a percent. SVGs also have advanced features, including filters, masks, CSS styling, color gradients, and interactivity; all of these except masks will be ignored for simplicity due to time constraints. An application of SVG color transformation is PDF color transformation: extract PDF pages as SVGs, apply transformations to them individually, and then recombine them into a single color transformed PDF.

2 Problem Description and Methods

In order to apply a color transformation to the SVG, the following steps are followed [4]:

1. find the bounding box of the SVG (width and height)
2. find a background rectangle or add one if there is none
3. find all `<mask>`s so they can be avoided in future steps.
4. find `stroke` colors outside of `<mask>`s, and apply the transformation to them
5. find `fill` colors outside of `<mask>`s, and apply the transformation to them
6. find embedded `<image>`s outside of `<mask>`s, and transform them using ImageMagick this only works if the transformation matrix is `[255 255 255; -1 -1 -1]` (inversion). it doesn't apply the transformation if the image id is referenced in a `<mask>`.
7. optimize `<path>` descriptors (optional)

The transformation goes as follows:

$\vec{A} = [a_r \;\; a_g \;\; a_b]$, $\vec{B} = [b_r \;\; b_g \;\; b_b]$, $\overrightarrow{C_{\text{in}}} = [r_{\text{in}} \; g_{\text{in}} \; b_{\text{in}}]$

$\overrightarrow{C_{\text{out}}} = \text{clamp}(\vec{A} + \vec{B} \odot \overrightarrow{C_{\text{in}}}, [0 \;\; 255])$

This takes $\vec{u} \odot \vec{v}$ as the Haddamard Product of $\vec{u}$ and $\vec{v}$ (`u .* v`)

For simplicity of input, the transformation can be passed as a single matrix:

$$M = \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} a_r & a_g & a_b \\ b_r & b_g & b_b \end{bmatrix}$$

For every color in the image, it has to first be converted to an RGB vector [5], then it can be transformed. After the transformation, it is converted to the shortest option between a hex code and named color. Alpha channels will not be changed on any color. colors that are "transparent", or "none" will also not be changed. Colors are found and manipulated through regular expressions (e.g. `regexp`).

2.1 Inputs and Outputs

The program output depends on the arguments given, it can print status updates if `"verbose"` is given, and it can print the output SVG content to a file, stdin, stdout, or nowhere, depending on the value of the `"outfile"` argument. Regardless of where it is set to output to, the content can still be acquired through the output arguments if so desired (e.g. `content = svg_color_tfm("outfile", "---");`). Only named arguments are accepted (e.g. `svg_color_tfm("infile", "./file.svg", "outfile", "-", "verbose", false);`). The accepted arguments are the following (all case insensitive):

```
"in", "infile"
    the input file path. should be string or char.
    defaults to "./in.svg".
"out", "outfile"
    the output file path. should be string or char.
    use "-" for stdout, "--" for stderr, or "---" for nowhere.
    defaults to the input file.
"verbose"
    whether or not to give progress updates through the console.
    should be a boolean. defaults to true.
"transformMatrix", "transform", "tfmat", "tfm", "M"
    should be a 2x3 double matrix.
    the top row is A and the bottom row is B.
    the color transformation is: outRGB = A + B .* inRGB.
    defaults to [255 255 255; -1 -1 -1] (color inversion).
"A"
    should be a 1x3 double matrix.
    only updates the top row of the transformation matrix.
"B"
    should be a 1x3 double matrix.
    only updates the bottom row of the transformation matrix.
"keepIntermediateFiles", "keepIntermediate", "keepInt", "keep"
    whether or not to keep temporary raster image files.
    should be a boolean. defaults to false.
```

```
"backgroundColor", "bgcolor"
    the background color before the transformation, if there isn't one.
    should be a string or char, and a valid color. defaults to "#fff"
"content"
    option to give the SVG content directly, rather than through a file.
    if both "content" and "infile" are given, the direct content is used.
    should be a string or char.
"help", "options", "-h", "-?", "-help", "--help"
    prints help text similar to this.
```

3 Test Cases

There are 314 test cases overall, 197 for global variables, 53 for utility functions, 50 for color functions, and 14 for the main functions [6]. All four test cases for the main function are included in the `svg` directory of the code folder [7], along with the example that will be shown here. The four examples all use color inversion (`M = [255 255 255; -1 -1 -1]`), but the main example `svg/main-test-in.svg` and `svg/main-test-out-%d.svg` use 20 different transformations that can be seen in the `main_test_transforms.m` file [7][8]. Most of these will be shown in this and the next page, and they were checked manually for accuracy. These examples represent the wide range of capabilities of this program. In the image captions, "tfm" means "transform". Transform 12 is a sepia-esque color transform, but due to limitations in the transformation matrix, a real sepia can't be constructed. This would require a 3x4 transformation matrix, rather than a 2x3 matrix. Transforms 3, 4, and 5 extract out a specific color channel, and transforms 8, 9, and 10 invert a specific channel. The test cases can be run either with `test_suite`, or with `test_gen("exec", true, "setup", "init_setup")` to regenerate and run them.

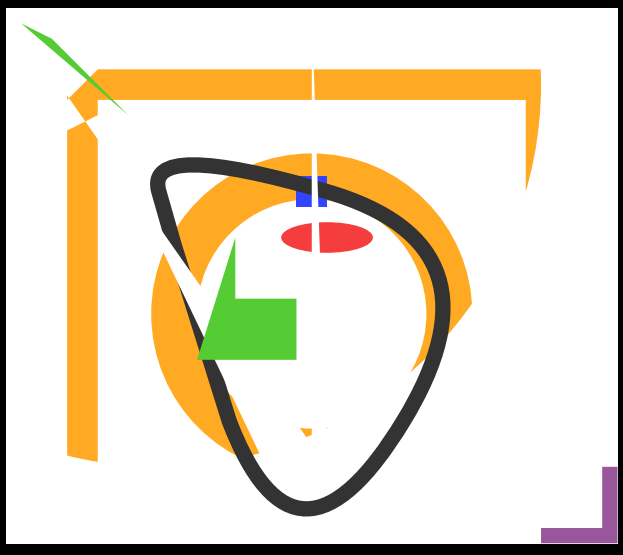

Figure 1: Original SVG (or tfm7)

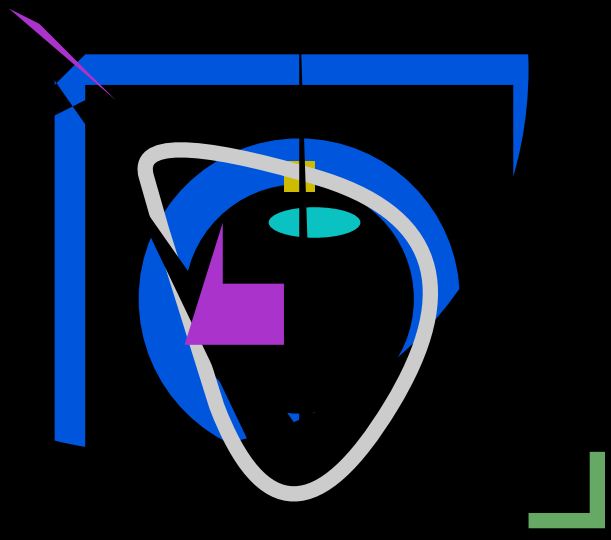

Figure 2: tfm1: Inverted SVG

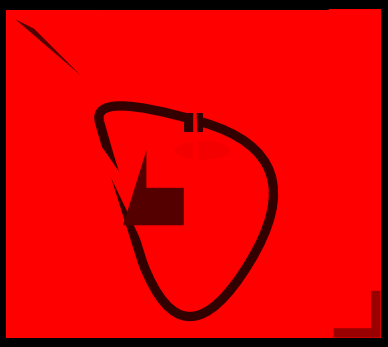

Figure 3: tfm3: red channel

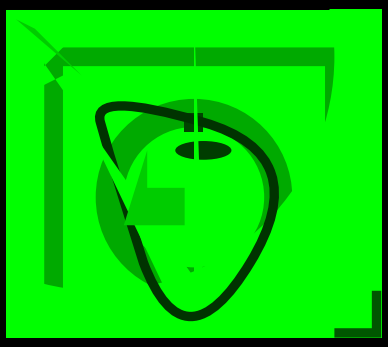

Figure 4: tfm4: green chan.

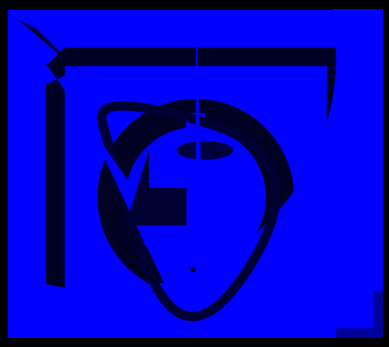

Figure 5: tfm5: blue chan.

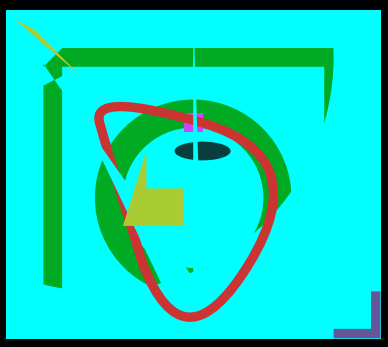

Figure 6: tfm8: inv. red

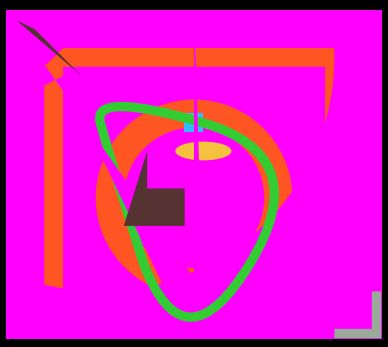

Figure 7: tfm9: inv. green

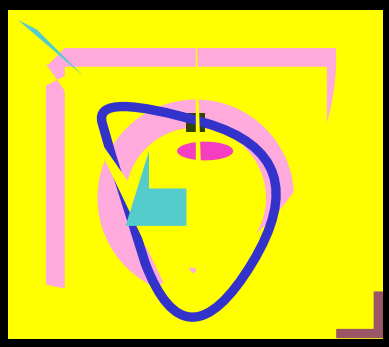

Figure 8: tfm10: inv. blue

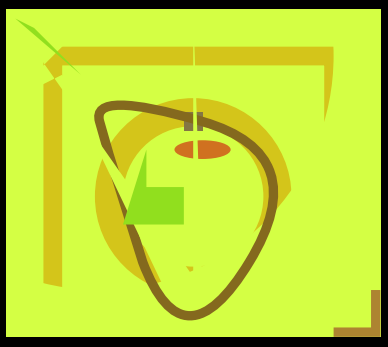

Figure 9: tfm12

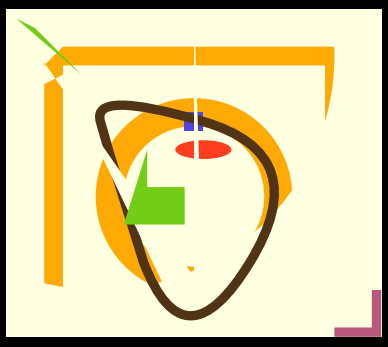

Figure 10: tfm15: cool filt.

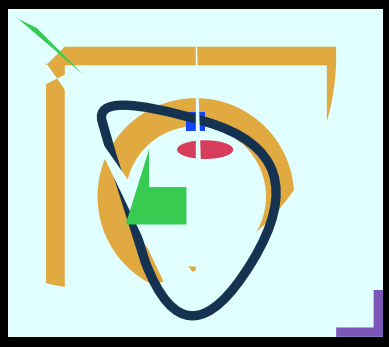

Figure 11: tfm16: warm filt

4 Major Code Sections

The following code segments are lightly edited for viewing ease, and for descriptive comments.

Snippet from `clr2hex`:

```
% convert rgba(R, G, B, A) to #RrGgBbAa
% as exemplified here, a regular expression approach is used for most things
matches = regexp(color, "^rgba\(" + ...
    strmul("\s*(\d+(?:\.\d+)?)\s*,", 4) + "{0}%?\)$", "tokens");
if ~isempty(matches)
    matches = str2double(string(matches{1}));
    if regexp(color, "%\s*\)$"), matches(4) = 255 * matches(4) / 100; end

    color = "#" + sprintf("%02x", round(matches));
end
```

Snippet from `apply_hex_tfm`:

```
% formula to go from '#RrGgBb' -> [r1 g1 b1] -> [r2 g2 b2] -> "#RrGgBb"
outhex = arrayfun(@(i) str2double(['0x' color(2*i + [0 1])]), 1:3); % hex -> double
outhex = uint8(A + B.*outhex); % apply tfm, clamp into [0, 255], and round.
outhex = "#" + sprintf("%02x", outhex); % uint8 -> hex
```

Snippet from `invert_fill_colors`:

```
% NOTE: the function's name is just leftover from a previous code version.
while true
    % find the next `fill` color and exit if there aren't any more.
    stt = indexOf(content, "fill=", iend);
    if isempty(stt), break, end

    counter = counter + 1;
    stt = stt + 5;

    % if the `fill` is inside a <mask>, don't transform the color.
    if any(median([ options.masks.maskRanges
            stt*ones(1, width(options.masks.maskRanges)) ]) == stt)

        iend = stt; % stop an infinite loop
        continue
    end

    iend = indexOf(content, """", stt + 1);

    % get the hex code of the transformed color
    color = apply_color_tfm( extractBetween(content, stt + 1, iend - 1), ...
        options.A, options.B);

    % update the SVG with the new color.
    content = extractBefore(content, stt + 1) + ...
        color + extractAfter(content, iend - 1);
end
```

5 References

[1] I. S. LLC, "ImageMagick," *ImageMagick*. https://imagemagick.org/script/index.php (accessed Dec. 07, 2024).

[2] E. Dahlström *et al.*, Eds., "Scalable Vector Graphics (SVG) 1.1 (Second Edition)," *W3C*, Aug. 16, 2011. https://www.w3.org/TR/SVG11/ (accessed Dec. 07, 2024).

[3] E. Dahlström *et al.*, Eds., "Basic Data Types and Interfaces – SVG 1.1 (Second Edition)," *W3C*, Aug. 16, 2011. https://www.w3.org/TR/SVG11/types.html#ColorKeywords (accessed Dec. 07, 2024).

[4] Daniel E. Janusch, "ERAU EGR 115 Final Project," *GitHub*, Dec. 08, 2024. https://github.com/drizzt536/files/tree/main/MATLAB/erau-egr115-final-project (accessed Dec. 08, 2024).

[5] Mohsen, "HSL to RGB color conversion," *Stack Overflow*, Feb. 28, 2010. https://stackoverflow.com/questions/2353211/hsl-to-rgb-color-conversion (accessed Nov. 16, 2024).

[6] Daniel E. Janusch, "/test_gen.m," *GitHub*, Dec. 08, 2024. https://github.com/drizzt536/files/blob/main/MATLAB/erau-egr115-final-project/test_gen.m (accessed Dec. 08, 2024).

[7] Daniel E. Janusch, "/svg," *GitHub*, Dec. 08, 2024. https://github.com/drizzt536/files/tree/main/MATLAB/erau-egr115-final-project/svg (accessed Dec. 08, 2024).

[8] Daniel E. Janusch, "/main_test_transforms.m," *GitHub*, Dec. 08, 2024. https://github.com/drizzt536/files/blob/main/MATLAB/erau-egr115-final-project/main_test_transforms.m (accessed Dec. 08, 2024).

6 Appendix – Pseudocode

```
define global variables
parse arguments
read input file, if applicable
find SVG height and width
look for rectangle that matches SVG height and width.
    add one if there wasn't one
find <mask> tags.
    find the indices of the starts and ends of the masks
    find the indices of embedded <image> tags
    find the ids of objects referenced in the mask
    store this information to avoid them in all later steps
find and apply transformation to all `stroke` colors
find and apply transformation to all `fill` colors
find all embedded <image> tags
    invert the image using ImageMagick.
    (this causes undefined behavior for M != [255 255 255; -1 -1 -1])
optimize the <path> descriptors in the svg
replace \r\n with \n and strip the svg, adding back one trailing newline.
print the content to wherever dictated by the "outfile" argument
set the output variable if applicable
```