Maldives Journal of Engineering and Technology
ISSN: 3007-6536 (Online)
https://mjet.mnu.edu.mv/

Algorithmic Optimization of Scalable Vector Graphics Through Structural Simplification and Transform Consolidation

Iyaan Azeez

[1]*Department of Computer Science, Faculty of Engineering, Science and Technology, The Maldives National University, Maldives;*

**Corresponding: s082605@student.mnu.edu.mv; iyaanazeez757@gmail.com;*

Abstract: Scalable Vector Graphics (SVG) files are widely used on the web due to them having smaller size compared to their raster counterparts. However, SVG files that are authored from vector suites might not have the most optimized representation. They contain significant data overhead and redundancies mostly arising from verbose metadata and non-optimized path definitions. To address these issues this work introduces a series of algorithms that target key areas that could yield size improvements. The algorithms work to decrease complex path definitions to much simpler representable SVG commands. To address attribute verbosity, the paper proposes a method that take in sequence of transformation commands and consolidate them into one single transformation matrix. Additionally, it also tackles embedded assets which have applied filters by pre-multiplying them ahead of time, allowing for the total removal of filter tags. After applying these method, the results have shown noticeable improvements of around 8.41% on some average. Although, the potential for compression and optimization depends highly on those files that are applicable to these transformations. This work has combined methods from areas such as computational geometry and rendering into SVG and has proved that there is room for improvement for more efficient representations within SVG without introducing a new vector image standard.

Keywords: Multimedia; Compression; Serialization; JSON; Schema-based

1. INTRODUCTION

Vector graphics represent images differently to raster images through parametric primitives such as points, shapes and paths. Compared to raster images they have a more structured representation and allow for much more compact storage making them ideal for applications such as user interfaces and is widely adopted on the web. They do not encode pixel information or their intensities. You define the structure of the image using paths and shapes. It is important to note that SVG is not a good fit for images with complex textures and photographs that can be converted to discrete vector instructions without converting each

Received: 20 February 2026
Accepted: 8 May 2026
Published: 28 May 2026

pixel of the image into a vector instruction which defeats the purpose of vector images. It is up to the implementer to render these instructions correctly providing flexibility.

1.1 Problem Statement

Traditional image formats such as PNG and JPEG are discrete representation models. This means they encode image information in discrete pixel units. Compression algorithms such as the *Discrete Cosine Transform* is able to exploit the human eyes inability to see colors in the high frequency domain and remove them. The quantization is achieved by first taking the pixel information and transforming them into the frequency domain. On the other hand vector graphics does not have a discrete unit. It uses a continuous representation model. It encodes the parameters of the function that will create the specific shapes and paths rather than sampling points from the function itself.

Due to this difference in representation models between vector and raster graphics it is not straightforward to apply traditional image compression techniques to SVG. All information is ultimately encoded as character byte array. These formats heavily rely on the relationship of neighboring pixels to determine how information can be quantized further. In normal raster images the smallest unit of information is encoded as a pixel. A pixel takes almost 3 bytes to encode the color information. This gives compression algorithms enough *entropy* to utilize. In these formats, compression is primarily a signal processing task. A compression algorithm will find the spatial correlation and leverage frequency transformations to mitigate inter-pixel redundancy.

$$H(X) := -\sum_{x \in X} p(x) \log p(x) \tag{1}$$

Due to the different nature of SVG compared to other formats makes it difficult to apply traditional image compression algorithms. Compressing SVG requires a paradigm shift. Instead of thinking of pixel density as the function to optimize we need to look for areas of topology and geometrical simplification. By optimizing the topology better we can reduce the byte count of the file.

2. LITERATURE REVIEW

Given the fundamental structural differences between vector based and rasterised images, conventional methods are not ineffective. Consequently, most research that focuses on optimizing vector images work on refining the *Document Object Model* of SVG. Arc elements represent the most significant contribution to the file size. To take an arbitrary Bézier curve B with n degree and having control points $B_0 \dots B_n$ and to sample points from it De Castelijau Algorithm [1] is used. The general form that the algorithm follows for a 3^{rd} degree is given as follows. Each dimension is evaluated separately. $b_{i,n}$ is the basis polynomial.

$$B_1(t)=\sum_{i=0}^{n} x_i b_{i,n}(t), t\in[0,1]$$
$$B_2(t)=\sum_{i=0}^{n} y_i b_{i,n}(t), t\in[0,1]$$
$$B_3(t)=\sum_{i=0}^{n} y_i b_{i,n}(t), t\in[0,1]$$

By performing a parametric evaluation of the curve it is possible to apply curve simplification algorithms. One of the more prominent algorithms that can be applied is Ramer-Douglas-Peucker algorithm [2] which take points of a Bézier curve of any arbitrary size and reduce it less points. It produces the most accurate generalization from a given set of points but can be time consuming. It is an iterative procedure that thins a set of points by discarding those that do not significantly contribute to the overall geometry based on a defined perpendicular distance threshold. While the standard recursive implementation of *RDP* can reach a worst-case time complexity of $O(n^2)$, more recent iterative solutions, such as those proposed by (Bi 2019) offer a more computationally efficient implementation. In [3], the research landscape of line simplification algorithms is explored in detail. It categorized the algorithmic approaches into 3 main groups.

- Methods such as the ***Nth point routine*** are computationally efficient and fast, as they eliminate points based on their sequence rather than their geometric relationship
- Algorithms like the ***Reumann-Witkam*** [4] routine and the *Lang* routine utilize a search region or tolerance bandwidth to evaluate relationships between consecutive points. The *Triangular routine*, inspired by McMaster's dissimilarity measurements, considers the areal displacement between the original and simplified poly-line, which helps preserve more geometric detail than simple distance-based local routines
- As mentioned above the ***Ramer-Douglas-Peucker*** algorithm is widely regarded as the mathematical benchmark for line simplification. It uses a divide-and-conquer approach to identify critical points that best represent the perceptual characteristics of the original line

SVG optimization research is heavily influenced by the needs of Geographic Information Systems *(GIS*) and cartography, where large-scale map data must be rendered efficiently in web browsers. In these fields, the concept of Level of Detail is critical. Objects further from the viewer are simplified more aggressively to reduce the workload on the graphics pipeline and improve frames-per-second *(FPS)* rates. Despite these advancements, [5] notes that the broader field of lossy SVG compression remains underdeveloped. While tools like *SVGO [6]* effectively handle "*lossy*" tasks, such as removing metadata and consolidating coordinate transforms, there is a notable lack of integrated solutions for the geometric "*overdefinition*" of complex paths or the pre-multiplication of embedded raster filters. This research seeks to bridge that gap by combining computational geometry with structural DOM refinement.

3. METHODS AND METHODOLOGY

In this section, we will elaborate in detail how the research discussed in literature review can be applied to our problem. It will discuss the line simplification algorithms and the composition of complex transformation matrices. Furthermore, it will discuss how we can discard unnecessary filters that embedded images are dependent on.

3.1 Structural Simplifications

One of the key-areas that needs to be focused on is the simplification of large shape types to its simpler variants namely, paths are the most versatile and text heavy tag in SVG. Paths cannot be categorized as primitive shapes such as rectangles, circles and lines. There were not many literature that was specifically discussing in a general sense of compression SVG files. Although, there were research being made into rendering of large GIS map data in the web using SVG in [7], [8]. Both literature discuss about converting and simplifying large path nodes to more simpler lines and later to Bézier curves. In the paper "*Approximation Of A Cubic Bézier Curve By Circular Arcs And Vice Versa*" by [9] gives a method that can approximate a circular arc to a set of cubic Bézier curves with an error of 1.96×10^{-4}. The error rate is a small penalty that would not be noticeable for our use cases.

Table 1: Characteristic of selected samples

Parameter	Description
x_1, y_1	The coordinates of the starting point on the line
r_x, r_y	The semi-major and semi-minor axes
ϕ	Angle from the *x*-axis of the current coordinate system to the *x*-axis of the ellipse
f_A	Large arc flag, 1 for arcs greater than 180
f_S	Sweep flag
x_2, y_2	Final coordinates of the arc

3.1.1 Coordinate Space Transformation

SVG stores the parameters of the Arc in the format given in Table 1 but, the method of approximation by [9] works with coordinates relative to the arc (*parametric equations*). The given parameters are normal coordinates in Cartesian space. Therefore, we need to do preliminary coordinate conversions. The SVG implementation provides equations that can convert from the endpoint coordinates of Arc command parameters to center parameterization [10]. After the coordinate transformations has been performed, the result will be a set of Cubic Bézier curves and its associated start, end and control points.

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \times \begin{bmatrix} \frac{x1+x2}{2} \\ \frac{y1+y2}{2} \end{bmatrix} \quad (2)$$

3.1.2 *Applying De Casteljau's Algorithm*

If the set of Cubic Bézier curves are minimal they would not require further simplification. Arcs of greater sizes might require a large amount of Bézier curves for an acceptable approximation. In that case a simplification of the points on the curve will need to be performed. To achieve this we can *De Casteljau's Algorithm* [1]. It is a recursive method to split Bézier curve to multiple smaller ones. With this algorithm we can sample a set of points from the curve. The algorithm provides a parameter to control the amount of sampled points. The variable $0 \le t \le 1$ controls the quantity of values. Higher value's will result in more sampled points giving a more accurate curve. You can see from Figure 1 how each segment $(p_0 p_1), (p_1 p_2), (p_2 p_3)$ get broken down according to $t = 0.25$

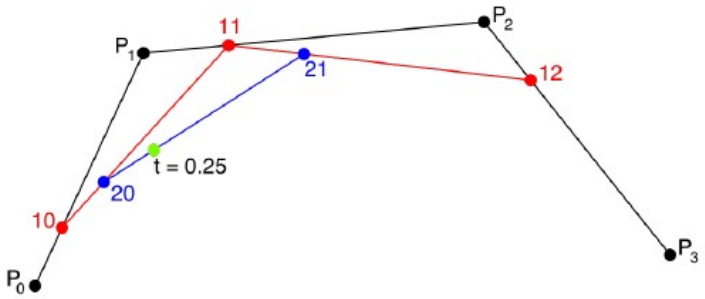


Figure 1: Subdivision of line segment in De Casteljau's [5]

The reason of applying De Casteljau's algorithm to get the sampled points was because it would be simpler to work with points of the curve rather than the polynomial form of the Bézier curve. Now we can apply a point simplification algorithm such as Ramer–Douglas–Peucker algorithm. It is a simple algorithm that discards extra points that do not contribute to the overall shape of the line. The first and last points is always kept and is not discarded. We then find the point that is furthest away from the segment of the first and last point. Use the perpendicular distance formula as we are working with normal coordinates. We use a defined constant ϵ as the threshold of which points can be discarded between the points. The default implementation provided in the reference literature is a recursive implementation. Given that we are working with n vertex points in the form (x, y) and there are $n-1$ segments. If we generalize the running time of the algorithm using the recurrence relation $T(n) = T(i+1) + T(n-i) + O(n)$. In the worst case it would run in $O(n^2)$ time. The reason is

that the algorithm explore all the points between the selected two points after finding the furthest point initially. There are iterative solution such as the one by [11].

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{3}$$

3.1.3 *Fitting Resulting Points To Bézier Curve*

As the final stage of this pipeline, we take the new simplified path points and fit them to a curve. The algorithm works by fitting a least squares line from a start and end points. This will approximate the tangents at the start and end points and give the general direction of the second and third control points. The in-between points lie in the neighborhood of the regression line. By using chord length parametization assign an initial parameter σ to each point d_i. Then calculate the distances α_1 and α_2 for the second and third points. After the control points are calculated you can now fit the curve to the control points.

3.2 Composition of Transforms

SVG allows elements to have specific transforms. Transform's can be used to change the orientation, position and scale of an element. Elements are arranged in a hierarchical order transforms applied to parent nodes will affect the relative transform of the children of that parent node. As of SVG 2, transforms can be applied to all the elements including the root `<svg>` element. Common applicable transformation commands are shown in Table 2. There are cases in SVG files where an element will have a composition of transforms defined in the attributes. In other words, a sequence of transforms. Below Figure 2 shows a scenario of chaining transformations. In terms of compression, this is a very verbose way to represent transformations. Vector authoring software has a tendency to group transforms together which can increase the file size. In this section we will look into how we can address this issue.

Table 2: Transform commands

Command	**Description**
translate(x, y)	Moves element in the *x,y* plane
scale(x, y)	Scale element in the horizontal or vertical direction
rotate(deg, [x], [y])	Rotates around *x,y* point in degrees
skewX(a)	Skew element in x axis by a degrees
skewY(a)	Skew element in y axis by a degrees

```
<g transform="rotate(10) translate(36 15.5) skewX(40) scale(1 0.5)">
```

Figure 2: Chaining transforms

3.2.1 *Representing Transforms as Matrices*

A more compact, less verbose way to represent multiple transformations is to use matrices. We can represent the coordinates of an element using a simple unit vector (x, y). The operation of applying a matrix transformation to a unit vector is a simple multiplication operation such as below. It shows a stretch matrix being applied to a set of coordinates. Assuming this operation will be performed for all the pixels that is within the boundary of an element. Assume k to be the scaling factor. $[x, y]$ is the column vector that represents the current coordinates of some arbitrary point of the element. $[x', y']$ represents the transformed coordinates. All of the transformations required by SVG can be represented with similar transformation matrices [12], [13].

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} k & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} = matrix(a, b, c, d, e, f)$$

3.2.2 Representing Transforms as Matrices

The latest SVG version supports a matrix command same as the commands mentioned in Table 2. Below shows a how a general transformation matrix can converted to work with the parameter structure of the $matrix$ command. We can apply any general transformation matrix to the current column vector coordinates of the element. Let x_n and y_n be the new coordinates after the transformation and x_p and y_p be the current coordinates before the transformation. Then it is a simple matrix multiplication. You can see that the coordinates of the element are represented in a 3D space column vector instead of a 2D column vector. That is because SVG and many other rendering application represent coordinates as homogeneous coordinates. Using homogeneous coordinates can simplify transformations and is a widely used concept in computer graphics [14]. One of the primary reasons we use homogeneous coordinates is because we cannot represent translation with a $2 x 2$ matrix. Moreover, it allows us to compose transformations into one singular matrix which we will be using to reduce the chain of transforms commands.

$$\begin{bmatrix} x_{new} \\ y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{prev} \\ y_{prev} \\ 1 \end{bmatrix} = \begin{bmatrix} ax_{prev} + cy_{prev} + e \\ bx_{prev} + dy_{prev} + f \\ 1 \end{bmatrix}$$

The transform commands in Figure 2 can be composed into the below single matrix. SVG specification clearly specifies that the transform attribute will apply the transformations from right-to-left. Below is the translation of the transform commands into its matrix equivalent and multiplying together to get the final transform that we can then turn into a matrix command within SVG.

$$rotate(10) \rightarrow translate(36, 15.5) \rightarrow skewX(40) \rightarrow scale(1, 0.5)$$

$$\begin{bmatrix} \cos 10 & -\sin 10 & 0 \\ \sin 10 & \cos 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 36 \\ 0 & 1 & 15.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \tan 40 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.98 & 0.32 & 32.58 \\ 0.17 & 0.56 & 21.45 \\ 0 & 0 & 1 \end{bmatrix}$$

3.3 Pre-Multiplied filters in embedded image

One of the most dominant issues with optimization of SVG for the web is handling of embedded raster image formats such as JPEG and PNG inside the structure. Mixing of raster and vector defeats the purpose of SVG but, some scenarios requires you to use specific assets that cannot be properly converted into a vector form. Such a case could be that you do not have the working files rather only the exported image. Image data is stored within SVG using Data URI and is encoded with *Base64* to ensure that binary data is converted to ASCII text and is readable. *Base64* is known for its verbosity and can produce larger file sizes due to it requiring multiple characters to encode a single byte.

Apart from the image data itself, if a user has edited the raster image within the vector image software by applying filters, these filters need to embedded into the SVG so that at the time of rendering, these effects can be applied on to the image. Now there is an opportunity to discard redundant information. Raster images already have the pixel data that will be rendered byte-to-byte without any difference. Therefore, it is possible to apply these filters to the pixel data beforehand so that we do not have to store filter tags. Figure 3 shows a snippet of a filter taken from an SVG file that has been authored using Inkscape.

```
<filter
    style="color-interpolation-filters:sRGB;"
    inkscape:label="Lightness-Contrast"
    id="filter33"
    x="0"
    y="0"
    width="1"
    height="1">
    <feColorMatrix
        values="0.829543 0 0 -0.181284 0.0852284 0 0.829543 0 -0.181284
0.0852284 0 0 0.829543 -0.181284 0.0852284 0 0 0 1 0"
    id="feColorMatrix33" />
</filter>
```

Figure 3: A lightness-contrast filter in Inkscape

3.3.1 Formulating Color Matrices

Color filters that modifies hue, saturation or any other color properties are mostly performed using a color matrix. The associated tag for the color matrix is the `<feColorMatrix>`. There are other tags that is used within the filter parent tag which corresponds to other filters such as Gaussian Blurs and operations which allows to blend between different filters and create composites. The paper will primarily discuss how a

lightness-filter can be pre-multiplied but, the same methodology can be applied to the other filters as well. The general format of the color matrix to be applied is shown below.

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} r1 & r2 & r3 & r4 & r5 \\ g1 & g2 & g3 & g4 & g5 \\ b1 & b2 & b3 & b4 & b5 \\ a1 & a2 & a3 & a4 & a5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

3.3.2 Pixel Level Execution Pipeline

To execute the pre-multiplication pipeline, the optimization tool must first isolate and decode the compressed binary data encapsulated within the Base64 string wrapper. The execution steps are structured as follows:

1. *Decompression.* The compressed container format is decompressed with the correct algorithm and unpacked into a sequence of raw pixel data.
2. *Channel Extraction.* The decoding engine loops sequentially or via vectorized lanes through the pixel matrix to isolate individual channel bytes.
3. *Multiply Filter Matrices.* The transformation matrix is applied directly to the extracted color coordinates, updating the raw bytes in place.

4. EXPERIMENTAL SETUP

In order to verify the impact and effect of the mentioned techniques in Section 3.1 3.2 3.3, benchmarks has been designed and performed. The benchmarks were conducted on a workstation powered by an Intel Core I5-10400 processor. This CPU features a base frequency of 2.90 GHz with a maximum turbo frequency of 4.30 GHz, supporting 12 threads. The cache architecture consists of 192 KB L1 cache with a 64B cache-line, a 1.5 MB L2 cache, and a 12 MB shared L3 cache along with 6 GB of DDR4 dual-channel memory operating at 2667 MT/s.

4.1 Dataset

Selecting a concentrated dataset that contains features that could be improved using the algorithms mentioned in this paper is critical. Otherwise the benchmarks will have zero effects. To ensure the evaluation framework is rigorous the dataset will be guided by the below criteria.

- *High Vertex Density.* Select SVG files with a high degree of coordinate character strings within their d="..." path attributes. Map data *(GIS)*, detailed icons, typography vector glyph, and complex line-art illustrations are ideal because they suffer from geometric "*over-definition*".

- *Presence of Elliptical Curves.* Ensuring the dataset contains a noticeable frequency of arc commands. This is mandatory to validate the coordinate space transformations and the circular arc cubic approximations outlined in your methodology.
- *Oversimplified Vector Geometry.* Avoiding datasets that are entirely composed of primitive shape commands and are therefore too simple.
- *Embedded Assets With Applied Filters.* Some samples must contain at-least one instance of an outside image asset with filters defined in the SVG file itself.

The samples for the dataset has been mainly sourced from *FreeSvg*. Although the dataset (N = 9) is limited, it tests our hypothesis. A larger dataset means we cannot focus on the details of the file structure as efficiently.

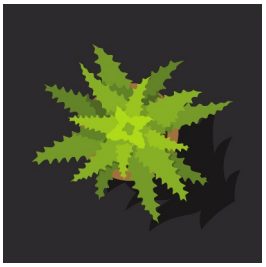
a) Sample 1

b) Sample 2

c) Sample 3

d) Sample 4

e) Sample 5



f) Sample 6

g) Sample 7

h) Sample 8

i) Sample 9

Figure 4: Full collection of the dataset

5. RESULTS

To evaluate the practical efficacy of the proposed structural simplification and transform consolidation frameworks, the algorithms were executed across a diverse profile of localized Scalable Vector Graphics *(SVG)* assets. The experimental test suite encompasses vector assets varying significantly in geometric complexity, node density, and absolute baseline scales. They range from lightweight graphic elements of under 10 KB to heavy, highly detailed compositions exceeding 3 MB. The performance metric that was captured was and primarily focused on was size related. The empirical data gathered from these optimization trials are structured and analyzed later establishing the boundaries of the methods discussed above.

Table 3: Effect to size after applying structural simplification

#	Name	Original (KB)	After (KB)	Reduction %
1	Football player 1b	131.7	129.2	1.90%
2	Jesus Christ depicted as the Good Shepherd	3100	2850	8.06%
3	A cute sheep	9	8.7	3.33%
4	Kebab skewer	24.5	22	10.20%
5	River in the valley	17.7	17.4	1.69%
6	Aloe Vera plant in the pot	122.5	119.3	2.61%
7	A cute squirrel	14.5	13.3	8.28%
8	Wind turbines	13.4	12.1	9.70%
9	Cupid on the Lion	570	543	4.74%

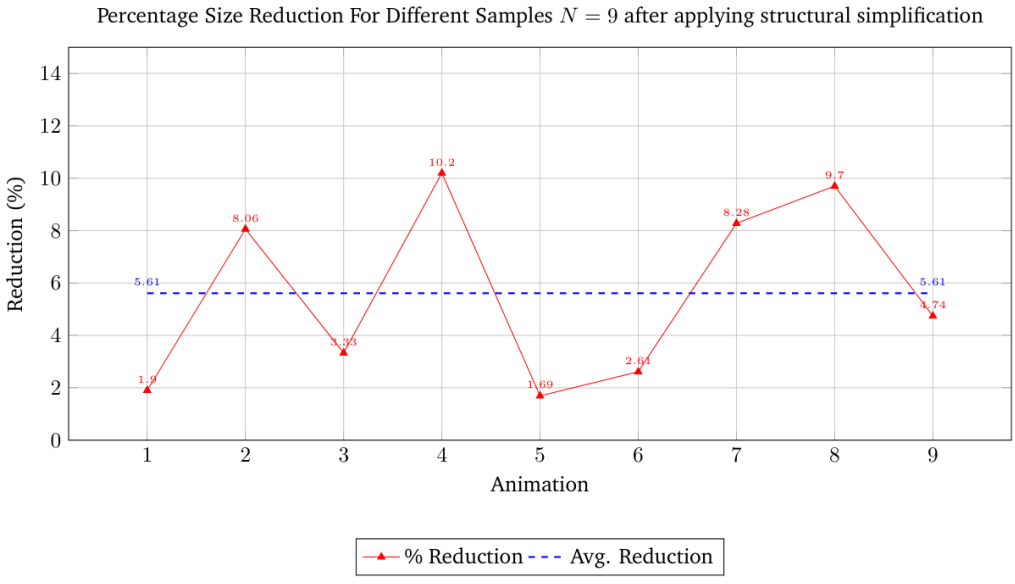


Figure 5: Trend of reduction after structural simplification

Table 4: Effect to size after composing transforms together

#	Name	Original (KB)	After (KB)	Reduction %
1	Football player 1b	131.7	129.97	1.31%
2	Jesus Christ depicted as the Good Shepherd	3100	3042.45	1.86%
3	A cute sheep	9	9.00	0.00%
4	Kebab skewer	24.5	24.06	1.78%
5	River in the valley	17.7	17.49	1.16%
6	Aloe vera plant in the pot	122.5	120.61	1.54%
7	A cute squirrel	14.5	14.29	1.48%
8	Wind turbines	13.4	13.40	0.00%
9	Cupid on the Lion	570	560.88	1.60%

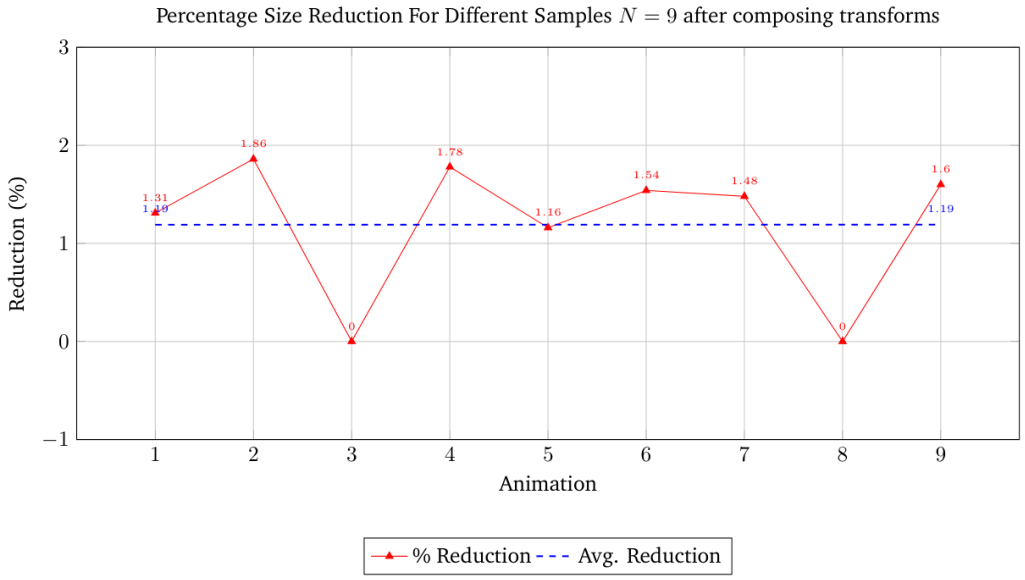


Figure 6: Trend of reduction after composing transforms

Table 5: Effect to size after pre-multiplying filters

#	Name	Original (KB)	After (KB)	Reduction %
1	Football player 1b	131.7	131.7	0.00%
2	Jesus Christ depicted as the Good Shepherd	3100	3100	0.00%
3	A cute sheep	9	9.00	0.00%
4	Kebab skewer	24.5	24.5	0.00%
5	River in the valley	17.7	16.2	8.47%
6	Aloe vera plant in the pot	122.5	122.5	0.00%
7	A cute squirrel	14.5	14.5	0.00%
8	Wind turbines	13.4	12.6	5.97%
9	Cupid on the Lion	570	570	0.00%

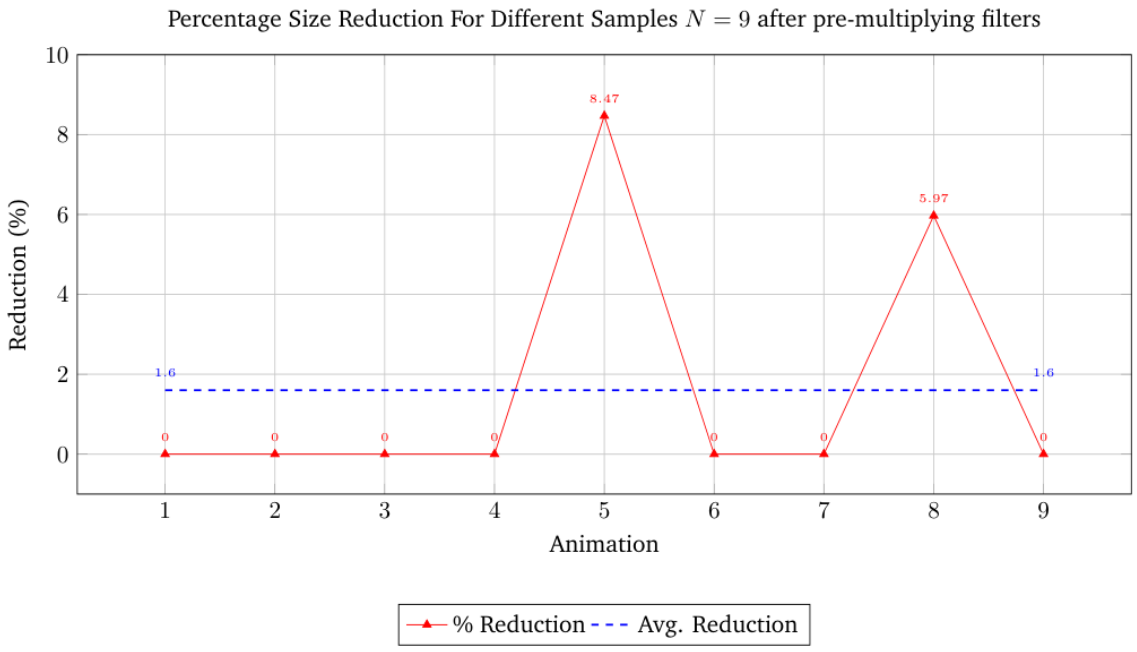


Figure 7: Trend of reduction after pre-multiplying filters

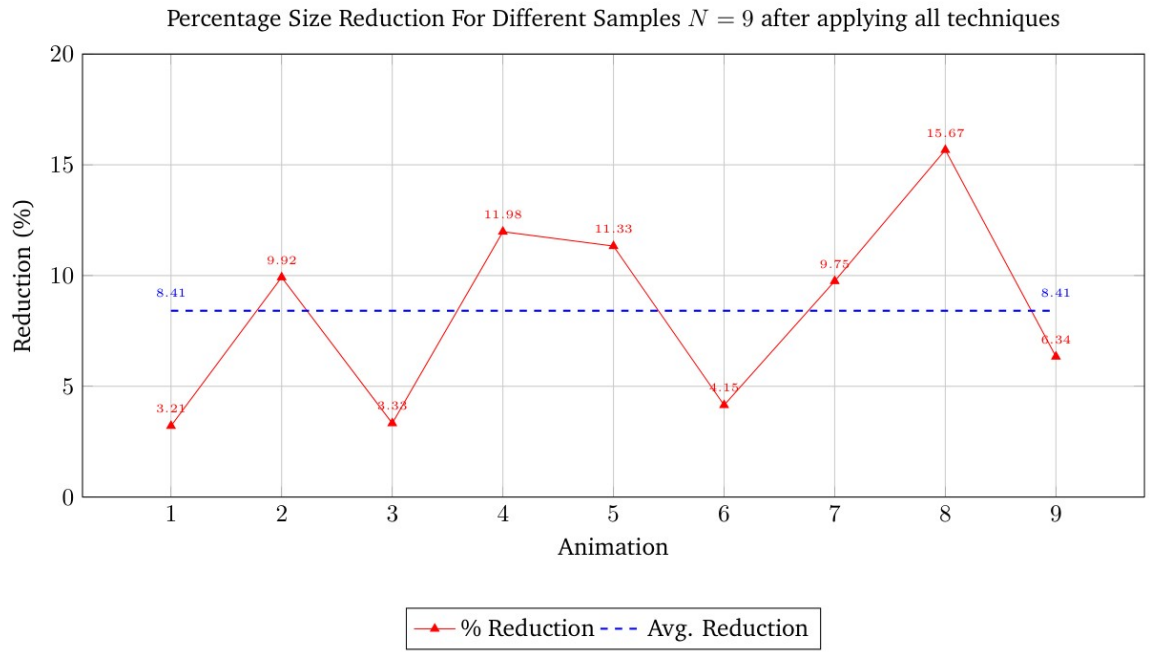


Figure 8: Trend of reduction after applying all 3 techniques

6. DISCUSSION

The results in Section 5 shows that the *SVG* optimization framework described in this paper is capable of delivering a reduction in file size. Although, the effectiveness of each individual technique is highly dependent on the characteristics of the input file. Results collectively indicate that meaningful reductions can be achieved without introducing visual degradation or requiring modifications to the SVG specification itself. If we look at Figure Error: Reference source not found, the average reduction is at about 8.41 % after applying all 3 techniques.

Among all the discussed techniques, *Structural Simplification* (Section 3.1) showed most substantial file reduction gains. As shown in Table 3 and Figure 5, the method achieved reductions ranging from approximately 1.69 % to 10.20 %, with an average reduction of about 5.61 % across the evaluated dataset. Variation comes from the fact that to the different geometric complexity factors of different samples. Files such as *Kebab skewer* and *Wind turbines* exhibited reductions of 10.20 % and 9.70 % respectively, indicating that these SVG contained path definitions or geometric structures that were highly suitable for simplification. Similarly, A cute squirrel achieved an 8.28 % reduction despite being relatively small in absolute size. This is more proof that optimization efficiency is not solely dependent on file size, but rather on the internal structural redundancy of the SVG document. Small files with inefficient path encoding may still contain significant optimization opportunities.

The reductions produced by *Transform Consolidation* (Section 3.2) were not as substantial as *Structural Simplification*, nonetheless it produced consistent results in the lower percentages. The average reduction was 1.19%. Unlike structural simplification, transform consolidation targets attribute verbosity rather than geometric redundancy. The results show that this method generally achieved reductions between 1% and 2%. Although this may appear modest compared to structural simplification, the optimization is computationally inexpensive and universally safe because it preserves the exact mathematical transformation. Two samples, A *Cute Sheep* and *Wind Turbines* $(S=\{3,8\})$, showed no measurable improvement from transform consolidation. This likely indicates that these SVG either contained no transform chains suitable for consolidation.

Filter Pre-Multiplying (Section 3.3) was applicable to a very small set of samples in the dataset. As shown in Table 5 and Figure 7, only two assets demonstrated measurable improvements after removing filter tags associated with embedded raster images. Samples that were applicable provided good results. The samples River in the valley and Wind turbines achieved reductions of 8.47% and 5.97% respectively, while all other samples showed no change. It important to note that strength of this technique is entirely dependent on whether the SVG contains embedded bitmap images with filter effects that can be pre-applied.

An important observation is that the methods operate on different layers of the SVG representation. Structural simplification targets geometric encoding, transform consolidation targets attribute representation, and filter pre-multiplication targets rendering metadata associated with embedded assets. Because these techniques address different forms of redundancy, they are complementary rather than mutually exclusive. This layered approach is particularly important for real-world optimization pipelines. It is a possibility for future research to add more techniques to this pipeline for more reduction.

Figure 10: Transformed by using a sequence of commands

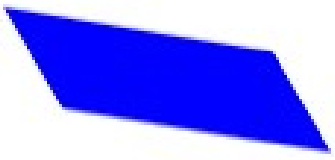


Figure 9: Transformed by using a single matrix

Figure 9, 10 shows how that using the matrix as a substitute does not change the outcome. Although, it require few characters to represent this transform compared to using the original method in Figure 2. You can observe from Figure 10 that all the rotations, skews and scaling has been applied properly. It is hard to see the translation effect from the diagrams. The change in color between the two shapes are just for the purpose of distinction and is not related to the transform.

7. CONCLUSION

Though the compression of SVG files is not novel, the methods outlined in this paper suggests that new methods can be applied. The majority of space in SVG is occupied by elements such as paths and arcs. The paper outlines methods such as the De Casteljaus Algorithm combined with Ramer-Douglas-Peucker algorithm can reduce the number of sampled points on the curve. The results shows that flattening of complex multi-transforms does not have an effect on the fidelity of SVG image and can reduce overall size of the image. This paper also present an idea on how the same approach of flattening transforms can be applied to filters for embedded raster images.

Further work and validation is required to determine the effectiveness of the method. It would also require testing against a more diverse image dataset. The next step for this research would be to compile these findings into a program that would be apply these methods in the real-world. The methods can be expanded outside of SVG as well into areas such as *Lottie-file* format which also deals with similar problems discussed in this paper. It would be interesting to further look into how machine learning can be applied into this problem as a predictor to find the most optimal representations of structures inside SVG.

Conflicts of Interests: The author declares no conflict of interest.

Data Availability Statement: The data supporting the findings of this study are available from the author upon reasonable request.

8. REFERENCE

[1] W. Boehm and A. Müller, "On de Casteljau's algorithm," *Comput. Aided Geom. Des.*, vol. 16, no. 7, pp. 587–605, Aug. 1999, doi: 10.1016/S0167-8396(99)00023-0.

[2] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Comput. Graph. Image Process.*, vol. 1, no. 3, pp. 244–256, Nov. 1972, doi: 10.1016/S0146-664X(72)80017-0.

[3] Ş. Ekdemir, "Efficient implementation of polyline simplification for large datasets and usability evaluation," Uppsala Universitet, 2011.

[4] K. Reumann and A. Witkam, "Optimizing Curve Segmentation in Computer Graphics. International Computing Symposium," *Ed Amst. North Holl.*, 1974.

[5] S. Ginn, "On the Compression of SVG Images," Universiteit van Amsterdam, 2017.

[6] L. Solntsev, "SVGO," SVGO. Accessed: Sep. 27, 2025. [Online]. Available: https://svgo.dev/

[7] S. Presnyakov, G. Boyarshinov, T. Borovskaya, and A. Rybkina, "Graphic File Formats for Web Virtual Globe," in *31th International Conference on Computer Graphic and Vision*, 2021, pp. 580–588.

[8] Z.-R. Peng and C. Zhang, "(PDF) The roles of geography markup language (GML), scalable vector graphics (SVG), and Web feature service (WFS) specifications in the

development of Internet geographic information systems (GIS)," *ResearchGate*, Aug. 2025, doi: 10.1007/s10109-004-0129-0.

[9] A. Riškus, "(PDF) Approximation of a cubic bezier curve by circular arcs and vice versa," *ResearchGate*, vol. 35, no. 4, Aug. 2025, Accessed: Sep. 22, 2025.

[10] "Scalable Vector Graphics (SVG) 2," W3C, Specification, Apr. 2018.

[11] R. Bi, *crdp (Cython implementation of Ramer-Douglas-Peucker algorithm)*. (2019). CPython. [Online]. Available: https://github.com/biran0079/crdp/blob/master/crdp.pyx

[12] R. C. Gonzalez and R. E. Woods, *Digital image processing*, Fourth, Global edition. New York, New York: Pearson Education, 2018.

[13] B. Jähne, *Digital image processing*, 6th rev. and ext. Ed. Berlin ; New York: Springer, 2005.

[14] R. Parslow, *Computer Graphics: Techniques and Applications*. in Computer Science. Springer US, 2013.