Evolutionary Image Vectorization

Matteo Destro (221222), *University of Trento*

***Abstract*—Vector graphic is a widely used way of representing images with a set of points connected by lines. Multiple domains require the usage of vector graphics, therefore an algorithm able to convert from raster to vector representations is essential: this process is called vectorization. This work proposes two approaches to vectorization, based on Genetic Algorithms and Particle Swarm Optimization, which try to encode the target image with a set of colored polygons or segments. Even if the final vectorized image is just an approximation of the target, the obtained results show that these methods can achieve good performances on this task and can be further extended to video vectorization.**

I. Introduction

VECTOR graphics are images defined in terms of points in a Cartesian plane connected by a set of lines or curves. This representation has several advantages over the more common raster graphics (where images are represented as a matrix of pixels), for instance a significant reduction in file size and the lack of aliasing when scaling up or down. These characteristics makes this representation the preferred one in many domains. However, raster graphics are still the dominant way of storing images, mainly because cameras capture light as a pixel matrix: an algorithm able to convert a raster image to its vector representation could therefore find a lot of applications in many fields. This particular task is called *vectorization*, and while its counterpart (i.e. *rasterization*, the conversion from vector to raster graphics) is rather straightforward, this problem is not trivial since often the complexity of an image makes it difficult to determine which details should be represented in the vectorized version. While multiple approaches to this problem can be found in the literature [1], [2], [3], the majority of them do not work well on all images, and very few approaches exploits evolutionary computation techniques [4].

This work proposes a vectorization method based on evolutionary algorithms. The main idea is to approach this task as an optimization problem that can be then solved using different metaheurisitcs. Two main approaches are explored, based on:

- *Genetic Algorithms*, where the image is represented through a set of colored polygons that are evolved to reproduce the target;
- *Particle Swarm Optimization*, where the objective is to reproduce the most relevant contours of the target using a set of lines (i.e. the particles).

In Section II and III the two approaches are explored and their implementation details are described, Section IV presents the setup of the experiments and the obtained results, while Section V discusses some of the limitations and possible further improvements of these algorithms.

II. Genetic Algorithm

In the *Genetic Algorithm* (GA) approach, the vectorization is performed by finding a set of polygons with different colors and transparencies that, when drawn together, produce an approximation of the target raster image. Each individual represents a possible vectorized candidate and is encoded by its set of polygons. Each polygon is composed by a set of vertices (each represented by its coordinates in the image space) and a RGBA color, as shown in Table I. The number of polygons in each individual can be either fixed or dynamic, based on the mutation operator used.

TABLE I
Genotype representation of an individual.

vertices	{(58, 12), …}	{(234, 721), …}	{(525, 851), …}
color	(128, 255, 121)	(145, 153, 248)	(92, 114, 194)
alpha	68	127	83

The objective of the genetic algorithm is to *minimize* a fitness function representing the loss between the target image and a vectorized candidate. The fitness is computed as the channel-wise, pixel-by-pixel, *sum of squared residuals* (SSR):

$$f(x) = \frac{1}{WHC} \sum_{i=0}^{W} \sum_{j=0}^{H} \sum_{c=0}^{C} (x_{ijc} - I_{ijc})^2 \tag{1}$$

where I is the target image and x_{ijc} is the value of pixel (i, j) and channel c of the individual rasterized representation x, computed by drawing its set of polygons over a black canvas, in order of appearance in the genotype.

The initial population is composed of individuals with randomly generated polygons. The vertices, colors and alphas of each polygon are sampled from a uniform distribution. A maximum distance between each vertex is used when sampling to favor the creation of smaller polygons.

At each generation a set of parents is selected from the population and recombined to generate a set of offspring. Different approaches were implemented and tested for parent selection, in particular: *roulette-wheel*, *rank-based*, *truncated rank-based* and *tournament selection*.

Similarly, for the crossover operator multiple methods were explored: *one point*, *uniform* and *arithmetic* crossover. The first two operates at the polygons level, i.e. only the polygons are recombined, while the vertex set of each polygon remains fixed. The latter method instead operates at the vertices level by computing the average between each vertex coordinate and color value for each pair of polygons. This method proved to be ineffective, since by computing the average the polygons tends to move to the centre of the image space, losing most of the details in the corners.

The mutation is done by applying a Gaussian noise to the polygons vertices and colors. Three different mutation step-sizes are used for, respectively, the vertices, color and alpha channel values. The mutation operators are devised to avoid generating invalid values: vertices that lie outside the image boundaries or channel values outside the $[0, 255]$ range are fixed. The algorithm can be configured to include a mutation operator able to add a randomly generated polygon to the current individual. Alternatively, it is possible to use an approach based on *Evolution Strategies* (ES) [5] to exploit self-adaptive mutation step-sizes: a set of strategy parameters used as step-sizes are added to the genotype and are evolved along with the solutions.

After the new offspring are generated, it is possible to use either a (μ, λ) or $(\mu + \lambda)$ replacement strategy to replace the old population. Section II-A presents an alternative replacement method for preserving the diversity of the solutions.

A comparison of the results obtained with different combinations of methods can be found in Table II.

A. *Niching mechanism*

After having implemented the solutions described previously and having run some tests, one major issue encountered was the lack of diversity between individuals after only a few generations. While to some extent this could be beneficial, since the problem has a very large search space and therefore we may want to maximize exploitation, the lack of diversity means that we are losing possible alternative good solutions early on during the evolution process and we are concentrating only on a small region of the search space, and therefore it may be beneficial to have a larger diversification between the solutions. To solve this problem, a *crowding* mechanism for survivor replacement was introduced, to improve the exploration and avoid issues of premature convergence.

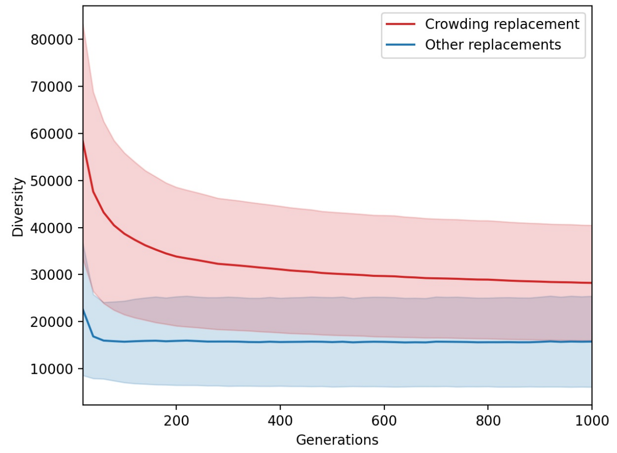

Fig. 1. Diversity results of the niching mechanism using crowding replacement. The diversity is computed as the sum of pairwise distances between each individual in a generation.

With this method, the members of the population are replaced one-at-a-time with each of the offspring: for each offspring, a random pool of individuals is sampled from the current population (the size of this pool is an hyper-parameter). From this pool, the individual that is the closest to the offspring currently analyzed is taken. If the current offspring's fitness is better than the closest individual, the new offspring replaces that individual in the population. It must be noted that it may be possible for one offspring to replace an earlier offspring in the same generation, given that the random sampled pool is taken from the current population (that could have already been updated).

Figure 1 shows a comparison of the evolution of population diversity with and without the niching mechanism, where the diversity value is computed as the summation of pairwise distances between each individual. Figure 2 shows a two-dimensional projection of the solutions genotypes: it can be seen how a clear separation of the population in different niches emerges.

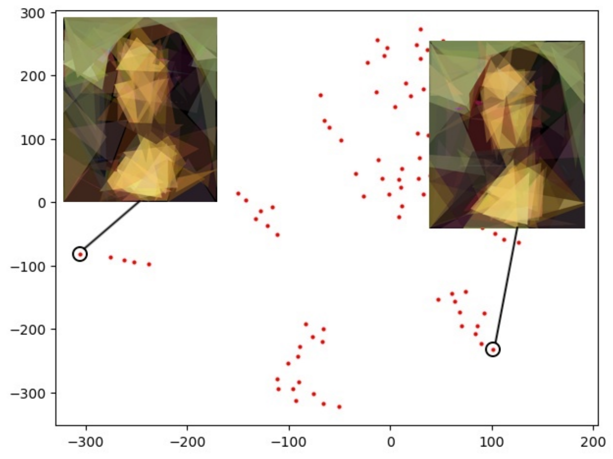

Fig. 2. Results of the niching mechanism using crowding replacement. The scatter plot is obtained by projecting the solutions genotypes into a 2D space using t-SNE[6].

III. Particle Swarm Optimization

The *Particle Swarm Optimization* (PSO) approach is significantly different from the previous one. In this case, the final goal is to obtain the set of lines that is most representative of the target image contours. It can be seen as an *edge detection* algorithm, but with the advantage of obtaining a vectorized representation of the edges (each line is represented by a pair of coordinates). Moreover, the search space is much more limited w.r.t. the genetic algorithm approach presented in Section II: in that case the number of dimension is $n_{\text{polygons}} \cdot (n_{\text{vertices}} + n_{\text{channels}}) \cong 50 \cdot (3 + 4) = 350$ at minimum, while with this method the search space is just three-dimensional. Each particle represents a fixed-length segment, encoded by a three-dimensional vector: two values for the segment's center coordinates and one for its rotation angle. This representation was preferred over a simple pair of coordinates to facilitate the movement of particles, without the need of an additional constraint to force the fixed segment size.

At each step, the velocity and position of each particle is updated as:

$$\begin{aligned} v' &= \mathrm{w} \cdot v + \phi_1 U_1 \cdot (y - x) + \phi_2 U_2 \cdot (z - x) \\ x' &= x + v' \end{aligned} \tag{2}$$

where w is the *inertia*, ϕ_1 and ϕ_2 are respectively the cognitive and *social coefficients*, U_1 and U_2 are random vectors sampled uniformly from $[0, 1]$, y is the personal best position and z is the neighborhood best position. The neighborhood of each particle can be either *distance-based* or *list-based* (*star* or *ring* topology). Other velocity update rules were also implemented and tested to preserve the diversity of the swarm and discourage premature convergence: *Fully-informed PSO* (FIPS) [7] and *Comprehensive Learning PSO* (CLPSO) [8].

The fitness of each particle is computed as the *gradient* of the grayscaled target image at the current particle position and in the direction orthogonal to the particle's rotation. The gradient is computed as the difference of intensity between pixels on the right and left sides of the particle's segment. A larger fitness corresponds to a larger gradient magnitude, i.e. a more relevant edge, therefore in this case the objective is to *maximize* the fitness. The fitness equation for a particle x is:

$$f(x) = \sum_{p_R \in R(x)} I(p_R) - \sum_{p_L \in L(x)} I(p_L) \tag{3}$$

where $I(p_R)$ is the pixel p_R intensity value and $R(x)$ and $L(x)$ are a set of points extracted respectively from the immediate right and left side of the particle's segment. Only a predefined number of points are sampled from the segment (i.e. 5 points for each side) to increase computation performances without affecting significantly the results.

To avoid very large velocity values, a *velocity clamping* mechanism is used to enforce a maximum velocity $v_{\max}$ s.t. $v_i' = \min(v_i', v_{\max})$.

A. Separation rule

An issue encountered was the fact that the particles tend to converge to a single local minimum, but in the case of edge detection we have many "optimum valleys" that we would like to cover homogeneously. To obtain the wanted result, a *separation vector* is computed and added to the velocity component before updating the particles positions. This separation component is inspired by the separation rule found in the Boids algorithm [9], and it is computed only for the position values, not for the rotation: the idea is that we want to maintain a minimum position distance, but we want particles that are near each other to align and have similar rotation values. The separation vector is computed as:

$$v_i = v_i - \sum_{p \in S_{d_{\min}};\, p \neq i} (x_p.\text{center} - x_i.\text{center}) \tag{4}$$

where $S_{d\min}$ is the set of particles with distance from i lower than a predefined threshold $d_{\min}$. By using this trick, it is possible to obtain a better separation and distribution of particles in the fitness landscape, avoiding the convergence into a single point.

IV. Experiments and Results

All the approaches presented were implemented in Python from scratch in order to maintain maximum flexibility. To further improve the computational performance, some key methods were compiled to native code using the *Numba* library. The final implementation is available in the linked GitHub repository[1].

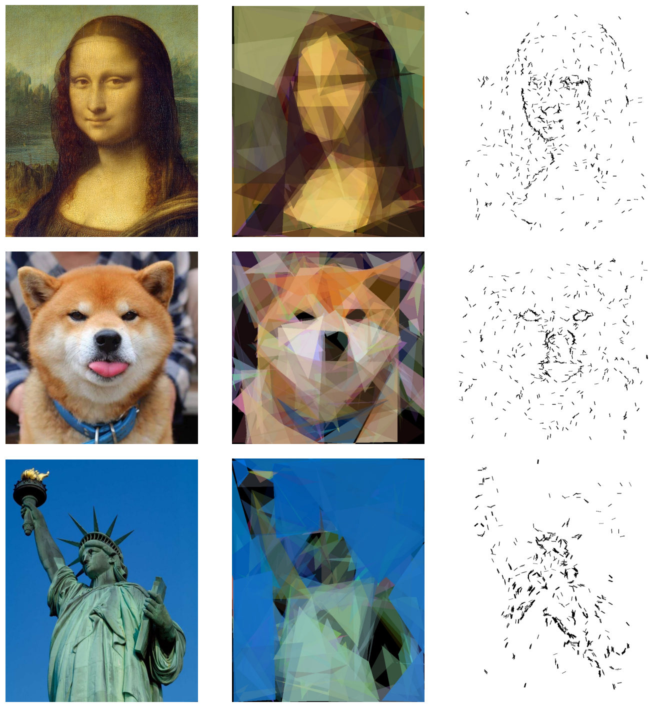

Fig. 3. A comparison of some results obtained by the GA and PSO approaches.

Different combination of hyper-parameters were tested, for both the GA and PSO approaches, for a total of 360 and 230 experiments respectively. Each combination was run for 1000 generations for GA and for 100 iterations for PSO. A comparison of the results obtained can be found in Table II and Table III. The results shows that both algorithms are able to extract a vectorized and meaningful representation of the target image, as it can be seen in Figure 3.

The reported results for GA were obtained using three vertices for each polygon: as shown in Figure 4, an higher number of vertices doesn't produce significant fitness improvements, however it drastically increases the execution time. Therefore only three vertices were used for the experiments.

The diversity mechanism implemented for the GA approach seems to be effective in maintaining a minimum diversity between the solutions in the population, as it can be seen in Figure 1 and Figure 2. However, by looking at the obtained fitnesses in Table II, it doesn't seems to provide significant improvements in the final results. Despite that, this mechanism proves to be still useful if the objective is to obtain multiple different solutions with one run of the algorithm.

For PSO, the standard velocity update rule appears to be more effective than other more complex methods (FIPS and CLPSO). Same goes for the minimum distance, a smaller value performs better than a larger one, probably because a large value does not allow for the particles to cover short contour lines.

[1]https://github.com/materight/evolutionary-image-vectorization

TABLE II
GENETIC ALGORITHMS RESULTS, USING 3 VERTICES FOR EACH POLYGON.

pop. size	**num. poly**	**selection**	**replacement**	**crossover**	**self adaptive**	**best fitness (%)**	**avg fitness**	**std fitness**
100	50	truncated(0.1)	(μ, λ)	uniform	False	422 (93%)	414.70	83.20
100	100	truncated(0.2)	crowding(2)	uniform	False	447 (93%)	425.61	49.51
100	200	tournament(10)	$(\mu + \lambda)$	uniform	False	455 (93%)	444.95	37.28
100	200	roulette-wheel	crowding(5)	uniform	False	458 (93%)	431.60	93.18
100	200	tournament(10)	(μ, λ)	uniform	True	513 (92%)	471.44	85.33
100	200	truncated(0.1)	crowding(5)	one-point	False	516 (92%)	509.82	18.51
50	100	truncated(0.1)	$(\mu + \lambda)$	uniform	True	517 (92%)	496.20	94.65
50	200	roulette-wheel	crowding(2)	uniform	False	524 (92%)	520.34	16.34
100	100	truncated(0.2)	(μ, λ)	one-point	False	524 (92%)	517.76	17.41
50	100	roulette-wheel	$(\mu + \lambda)$	uniform	False	526 (92%)	519.46	16.36

TABLE III
PARTICLE SWARM OPTIMIZATION RESULTS.

swarm size	**segment size**	**velocity update**	**neigh. topology**	**neigh. size**	(w, ϕ_1, ϕ_2)	$d_{\min}$	$v_{\max}$	**best fitness**	**avg fitness**	**std fitness**
1000	10	standard	distance	5	(0.1, 1.7, 1.2)	2	50	67.94	66.57	3.75
500	20	standard	star	5	(0.1, 1.7, 1.2)	2	20	65.83	62.87	8.66
500	10	standard	star	5	(0.1, 1.7, 1.2)	2	50	62.21	62.14	0.19
1000	10	standard	star	3	(0.1, 1.7, 1.2)	2	50	57.29	54.86	9.04
1000	20	FIPS	star	3	(0.7, 1.5, 1.5)	2	50	44.65	42.12	8.73
1000	20	standard	distance	3	(0.1, 1.7, 1.2)	2	50	42.70	40.92	7.28
500	20	FIPS	distance	5	(0.1, 1.7, 1.2)	2	50	41.87	37.13	9.79
500	10	CLPSO	ring	5	(0.1, 1.7, 1.2)	2	20	32.49	31.11	3.19
1000	20	standard	ring	5	(0.1, 1.7, 1.2)	10	20	22.35	19.83	8.98
500	10	CLPSO	star	5	(0.1, 1.7, 1.2)	2	50	21.92	20.70	3.74

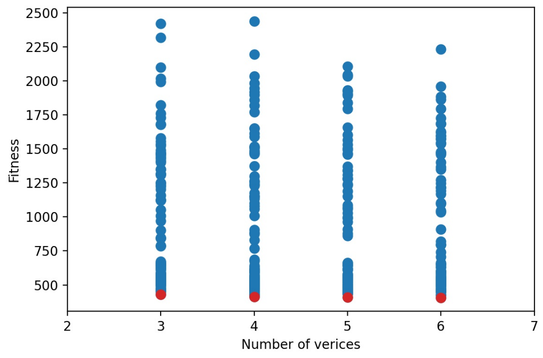

Fig. 4. Results of the fitnesses obtained using different number of vertices for each polygon.

V. CONCLUSIONS

In this work two methods for image vectorization were proposed, based on genetic algorithms and particle swarm optimization. The obtained results shows that even if the final generated vectorized graphics are just an approximation of the target images, these methods are both effective in reproducing the most relevant visual features.

These methods can be also extended beyond the scope of static images, for instance they can provide an efficient approach to *video vectorization*. The main idea is that it is possible to optimize over the first video frame for a significant number of generations, like in the case of images. Then for the successive frames the algorithm can run just for a limited number of generations, starting from the result obtained in the previous frame. This can be done under the assumption that the represented objects make small movements between frames, and therefore the vectorization of the previous frame is a good starting base for vectorizing the current one.

REFERENCES

[1] J. Sun, L. Liang, F. Wen, and H.-Y. Shum, “Image vectorization using optimized gradient meshes,” *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 11–es, 2007.

[2] T. Xia, B. Liao, and Y. Yu, “Patch-based image vectorization with automatic curvilinear feature alignment,” *ACM Transactions on Graphics (TOG)*, vol. 28, no. 5, pp. 1–10, 2009.

[3] J. Jimenez and J. L. Navalon, “Some experiments in image vectorization,” *IBM Journal of research and Development*, vol. 26, no. 6, pp. 724–734, 1982.

[4] S. Bergen, “Evolving stylized images using a user-interactive genetic algorithm,” 01 2009, pp. 2745–2752.

[5] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, “A survey of evolution strategies.” 01 1991, pp. 2–9.

[6] L. van der Maaten and G. Hinton, “Viualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[7] R. Mendes, J. Kennedy, and J. Neves, “The fully imformed particle swarm: Simpler, mabe better,” *Evolutionary Computation, IEEE Transactions on*, vol. 8, pp. 204 – 210, 07 2004.

[8] J. Liang, K. Qin, P. Suganthan, and B. Subramanian, “Comprehensive learning particle swarm optimiser for global optimisation of multimodal functions,” *Evolutionary Computation, IEEE Transactions on*, vol. 10, pp. 281 – 295, 07 2006.

[9] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” in *SIGGRAPH ’87*, 1987.