Image Approximation via Semi-Transparent Shape Evolution

Matteo Liotta
SM3800072
Optimization for Artificial Intelligence, 2025, UniTS - *Final Project*

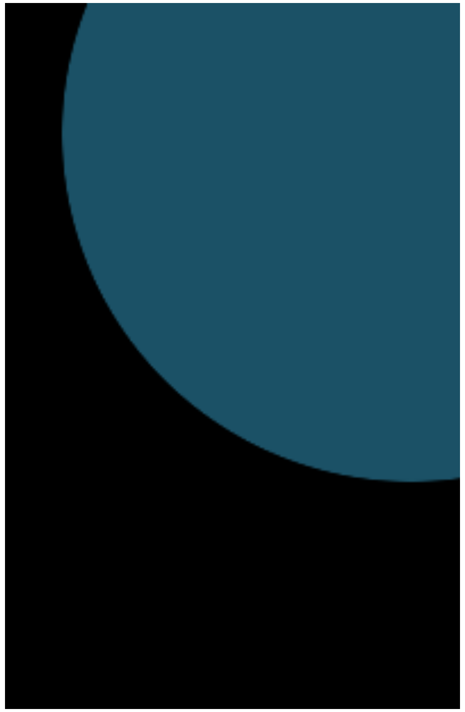
Problem Introduction

Problem Introduction

This project is a revisit of the "Genetic-lisa" Peter Braden's project, which aims to recreate images with genetic algorithms.

It had different implementations over time, from Python to C

The base concept is to evolve a series of overlapping shapes to reach a certain level of *similitude* with the original picture.



Problem Introduction

Similitude which is measured with defined objective function

$$\sum_{\substack{i=1,\dots,W\\ j=1,\dots,H}} \left({r_{i,j}}_{true} - \widehat{r_{i,j}}\right)^2 + \left({g_{i,j}}_{true} - \widehat{g_{i,j}}\right)^2 + \left({b_{i,j}}_{true} - \widehat{b_{i,j}}\right)^2$$

How are individuals defined?

How are individuals defined?

Shapes

Color

How are individuals defined?
Shape Definition

In the original code different shapes are proposed, but here we can focus on

- Ellipses
- Rectangle
- Triangles

defined as a dictionary of **name** and **coordinates** genetated at random

```
{'type': 'triangle', 'coords': [44, 213, 93, 148, 163, 220]}
```

Shape Geometry Definition

```
{'type': 'triangle', 'coords': [44, 213, 93, 148, 163, 220]}
```

(x,y) (x,y) (x,y)

y y + shape height

```
{'type': 'rectangle', 'coords': [31, 136, 48, 160]}
{'type': ‘ellipse', 'coords': [31, 136, 48, 160]}
```

x x + shape width

Shape Color Definition

Also color (A,R,G,B) tuples are generated at random

```
def generate_color(prev = None, a_mutation = (-0x5,0x3), rgb_mutation = (-0x5,0x3)):
    if prev:
        return (prev[0] + random.randint(*a_mutation),  # R
                prev[1] + random.randint(*rgb_mutation),  # G
                prev[2] + random.randint(*rgb_mutation),  # B
                prev[3] + random.randint(*rgb_mutation))  # A

    return (random.randint(0x33, 0x99), # R
            random.randint(0, 0xff), # G: between 0 and 255
            random.randint(0, 0xff),     # B: between 0 and 255
            random.randint(0, 0xff))     # A: between 0 and 255
```

We'll see this later…

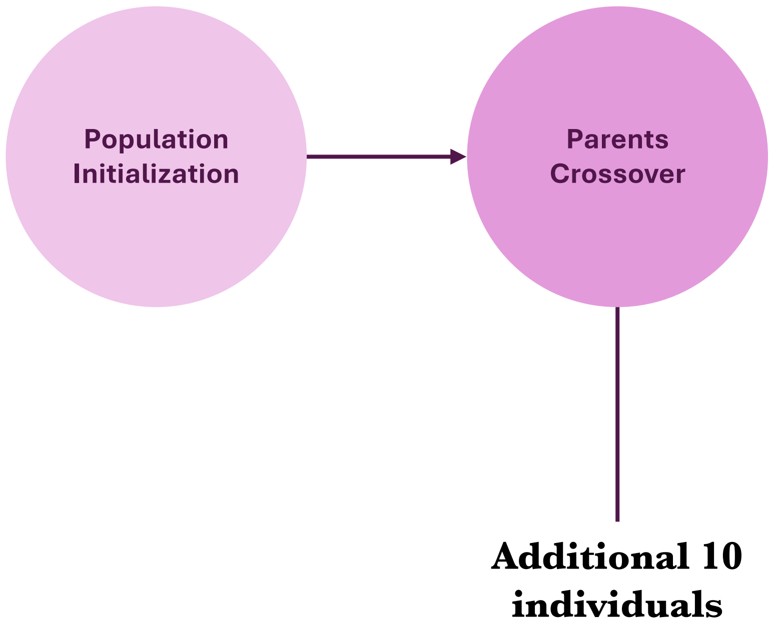
Individuals

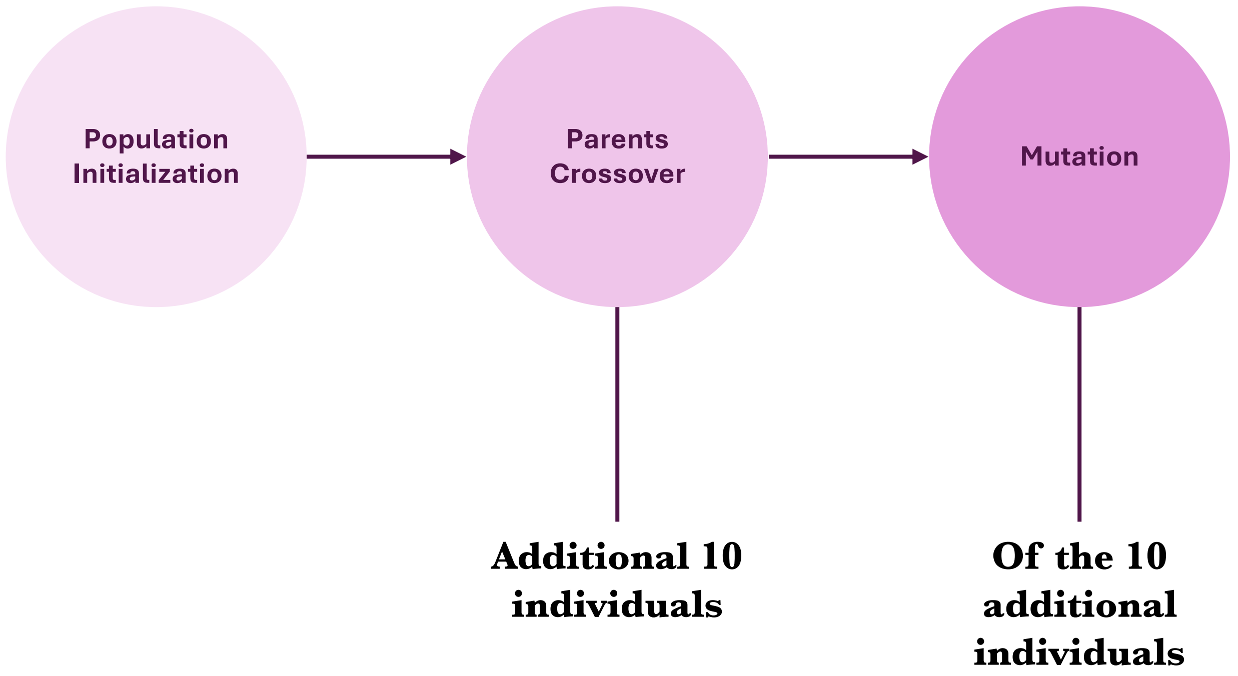
Name	Dna	Fitness value
String	*{shapes: {type, coordinates}, colors : color tuple}*	*Scalar*

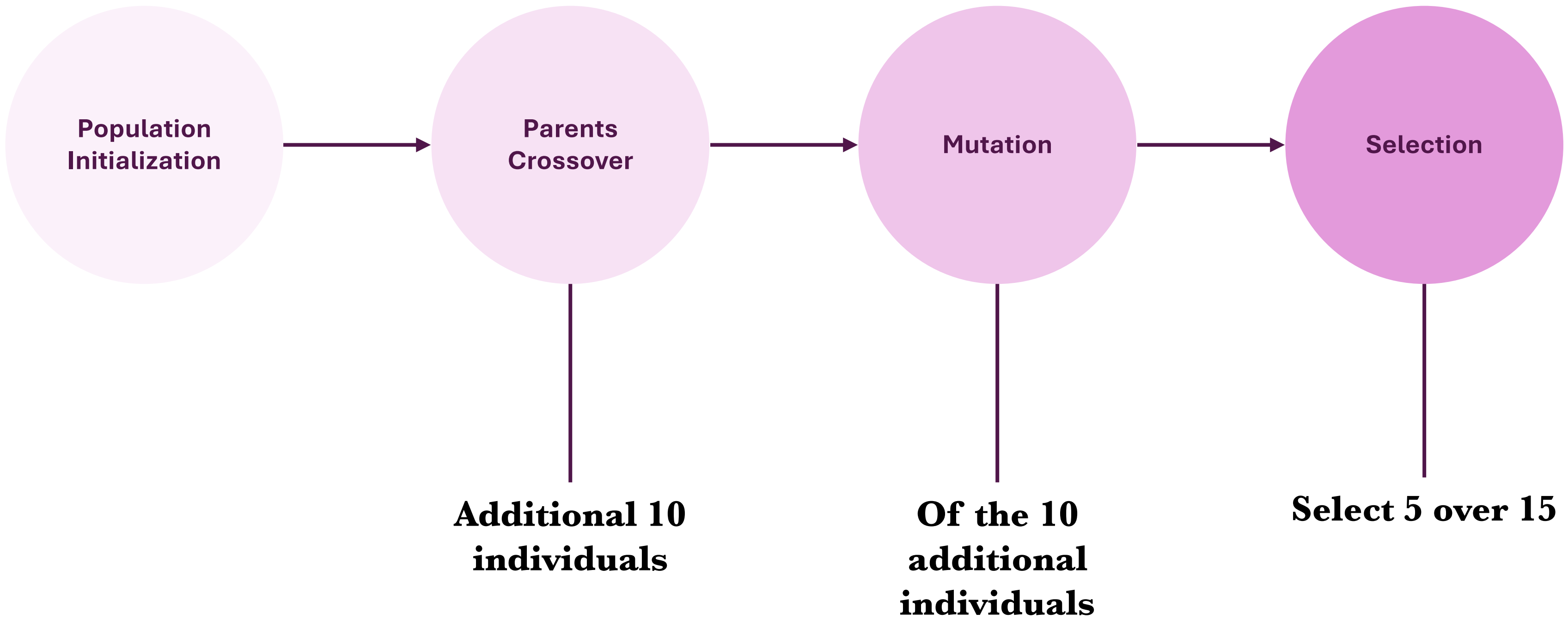
Evolutionary cycle

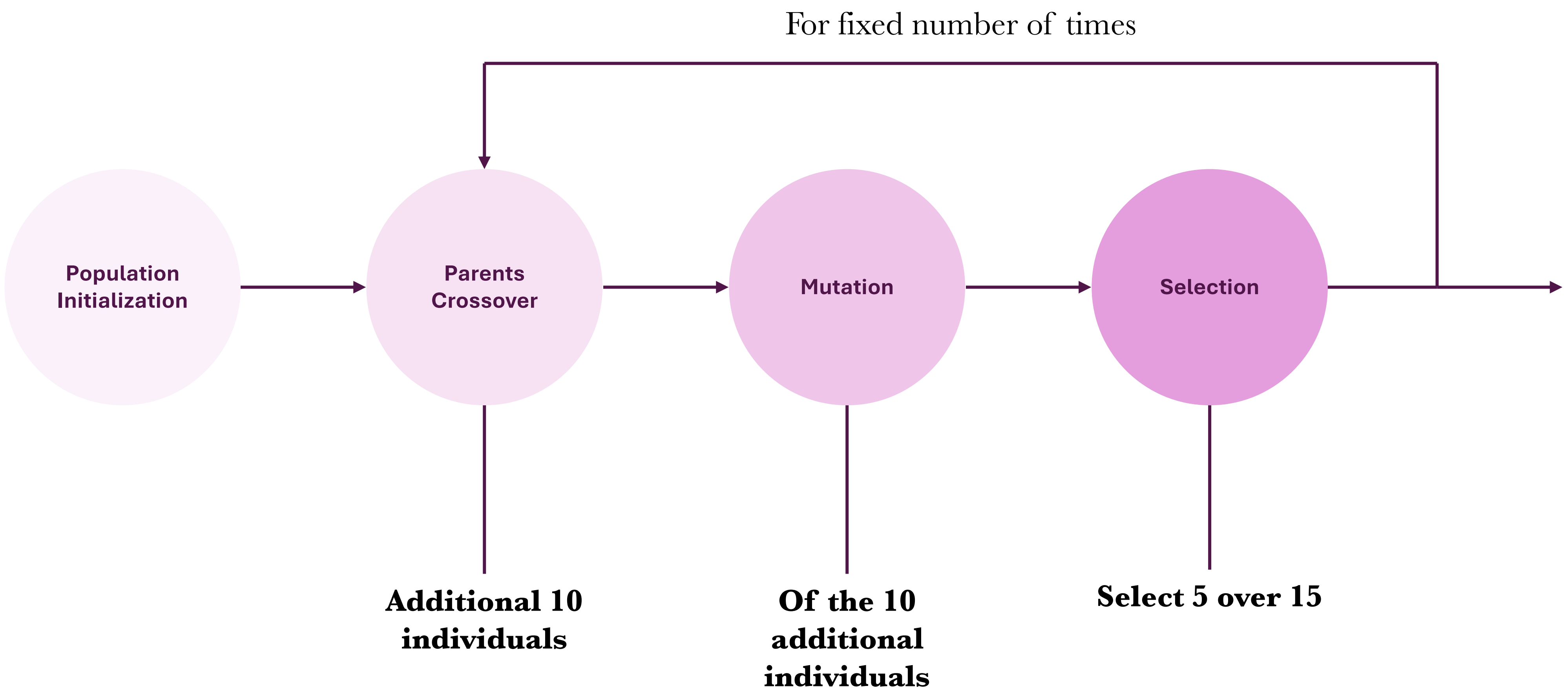
* By original proposed code

Population
Initialization


Population
Initialization
Parents
Crossover
Additional 10
individuals


Population Initialization
Parents Crossover
Mutation
Additional 10 individuals
Of the 10 additional individuals


Population Initialization
Parents Crossover
Mutation
Selection
Additional 10 individuals
Of the 10 additional individuals
Select 5 over 15

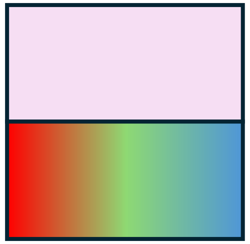

For fixed number of times
Population Initialization
Parents Crossover
Mutation
Selection
Additional 10 individuals
Of the 10 additional individuals
Select 5 over 15

How is crossover defined?

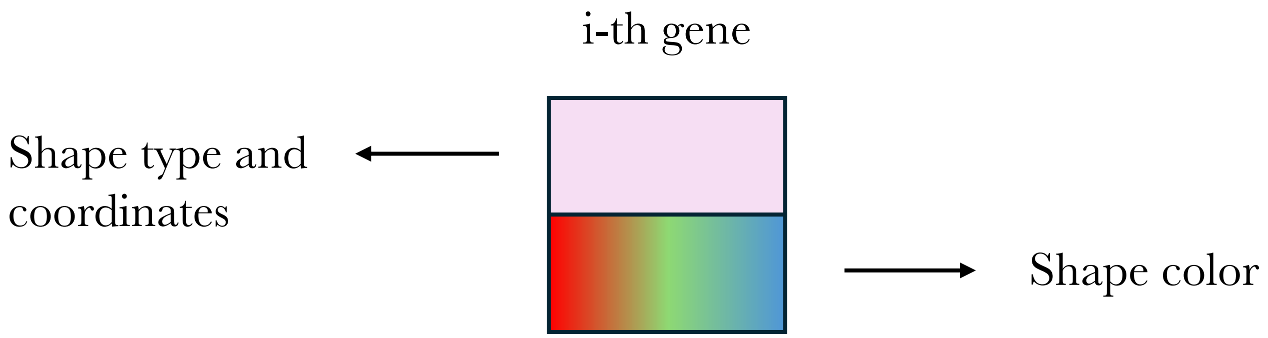
Individual Recap

Each individual has its own Dna, defined of atomic units: the genes

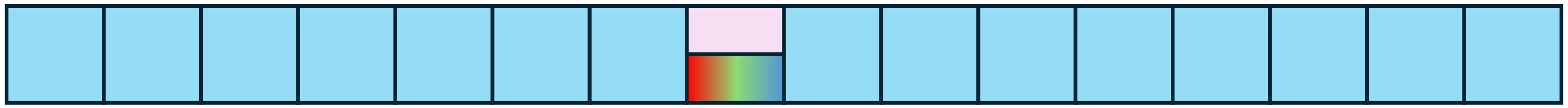
i-th gene



Individual Recap



Individual Recap



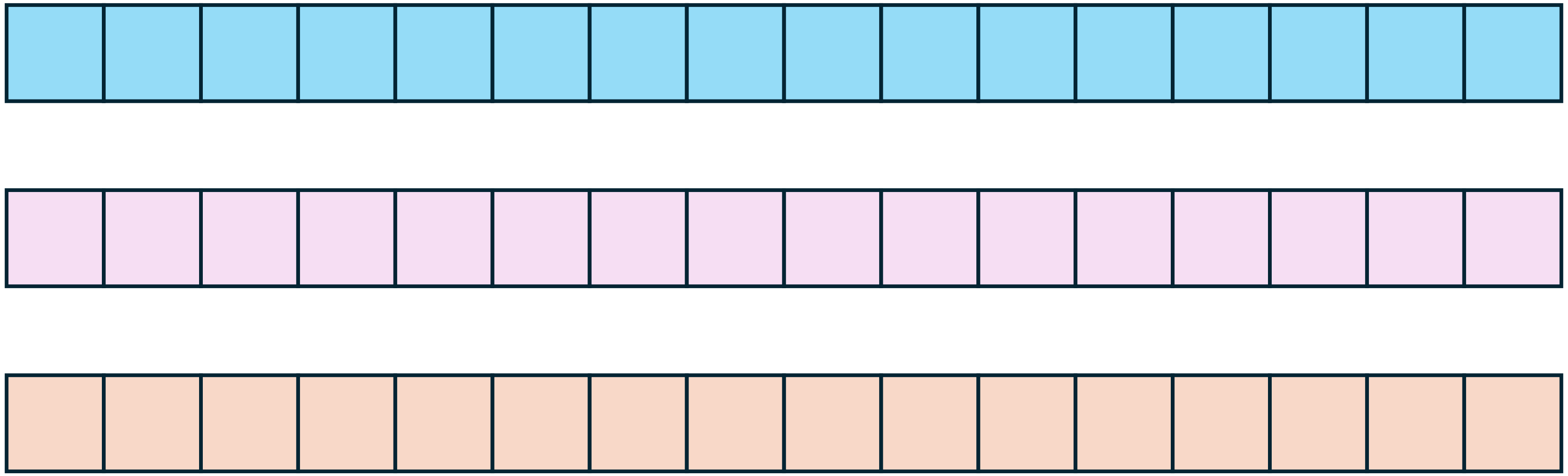
How is crossover defined?

We start with a parent population

Parent 1

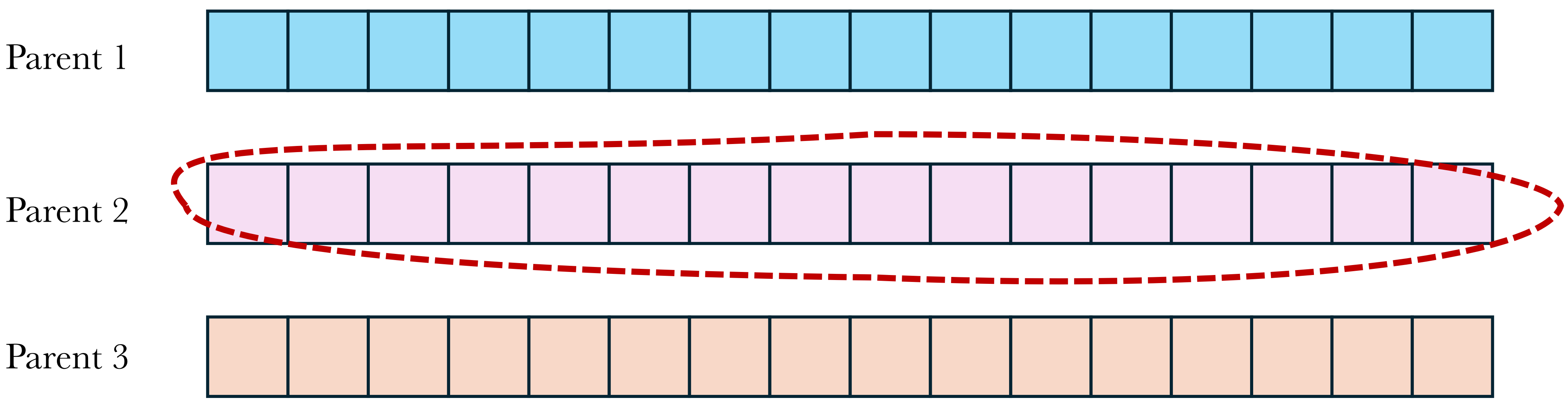
Parent 2

Parent 3



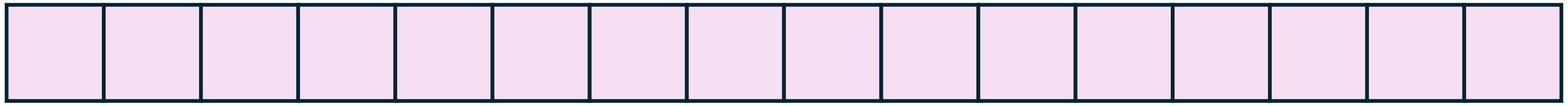
How is crossover defined?

To generate a new individual we select **randomly** a **parent dna and deepcopy it**.



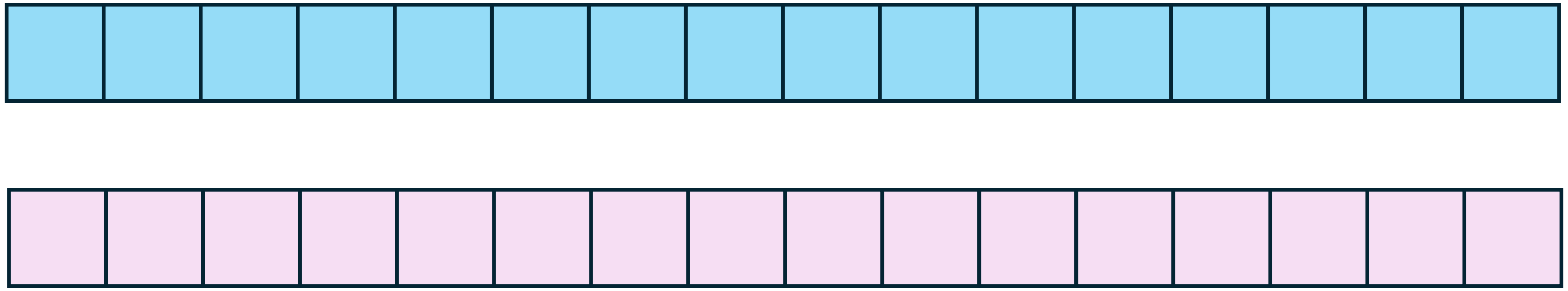
How is crossover defined?

Then iterate on parents: **change one random gene of in place of another of this offspring**



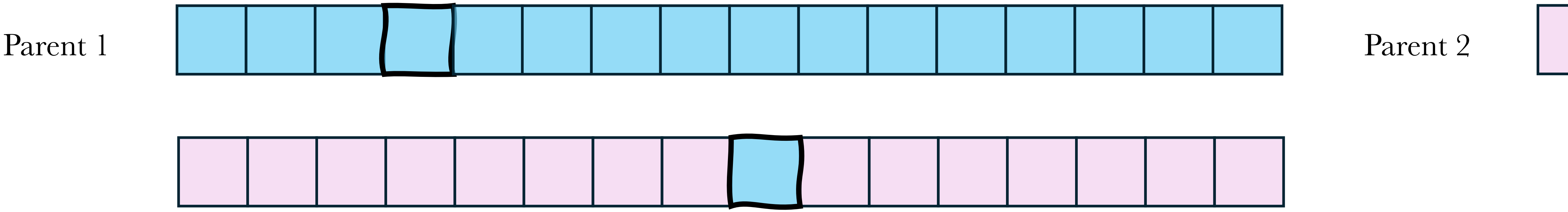
How is crossover defined?

Parent 1

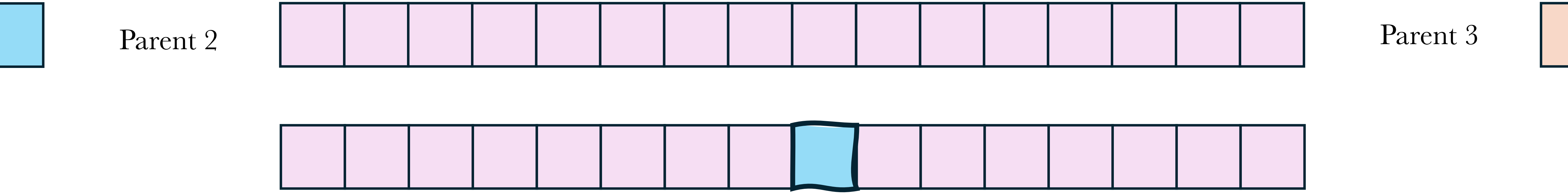


Parent 2

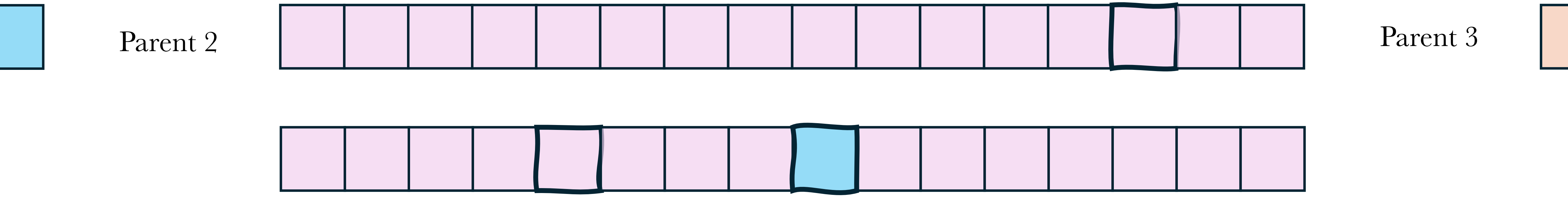
How is crossover defined?



How is crossover defined?

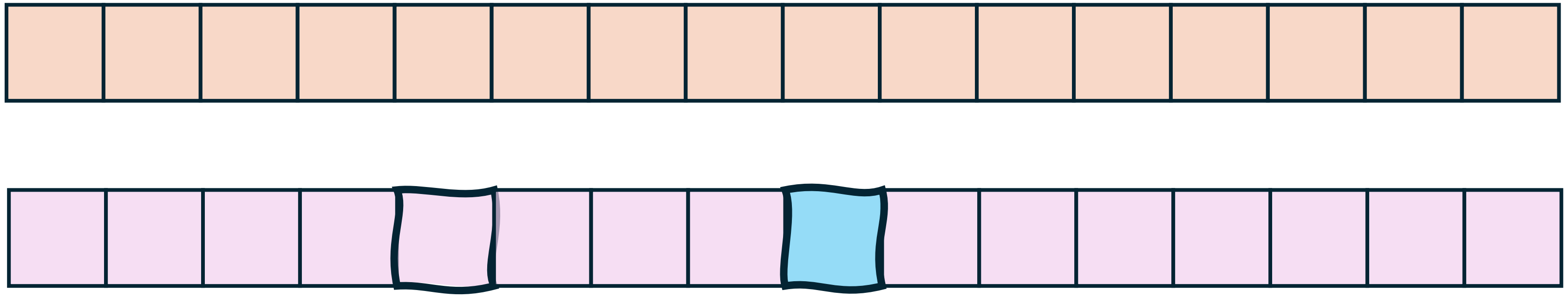


How is crossover defined?

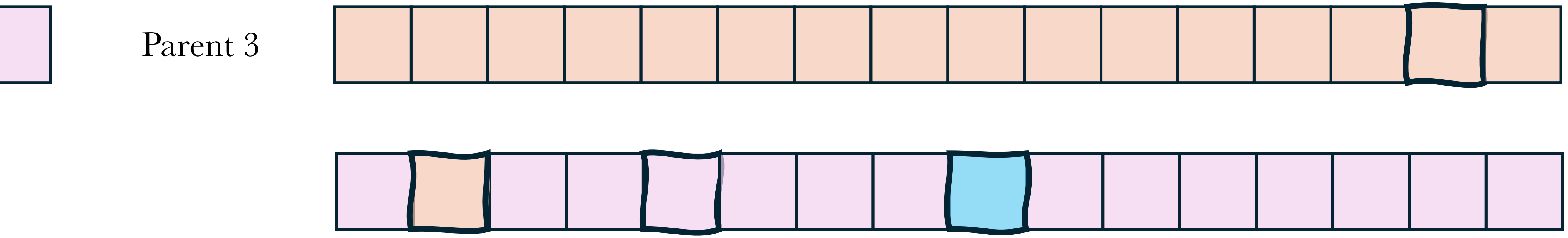


How is crossover defined?

Parent 3

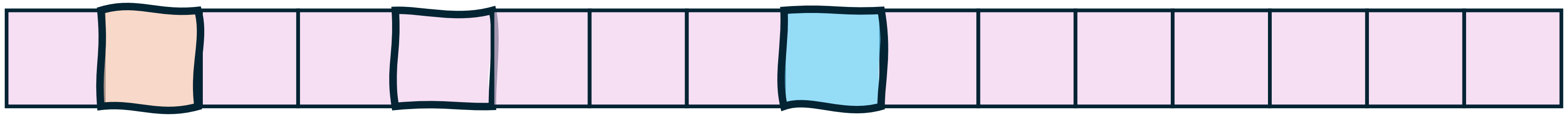


How is crossover defined?



How is crossover defined?

After that, mutation will occur to each new offspring.



How is mutation defined?

Individuals

Three different actions:

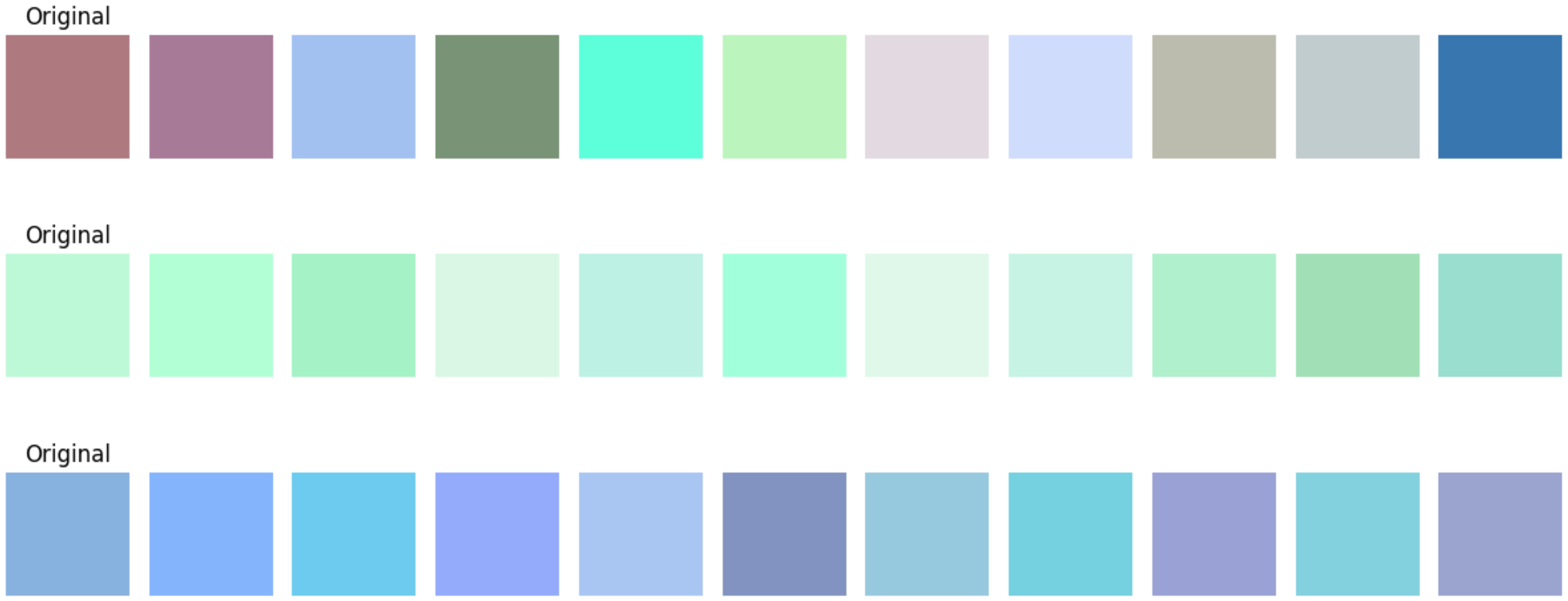
- New random shape **appending** to offsprings DNA → ***len<500 and 25% probability***
- Shape change and color **change** of a gene → ***elif 75% probability***
- Random shape **removal** to offspring DNA → ***else***

But… the change function can now handle a new feature: we can change **not by deletion** and **random reinsertion** of a gene, but deletion and **conditionate re-generation**.

Shape Color “Prev” Modification

Using additional parameters we can *eventually* mutate the color avoiding random re-generation

⇒ We can eventually avoid knowledge discard and change in the around.



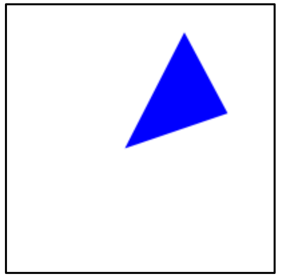
$\longrightarrow$ *Original Code*

$\longrightarrow$ $a \pm 0, \text{rgb} \pm 50$

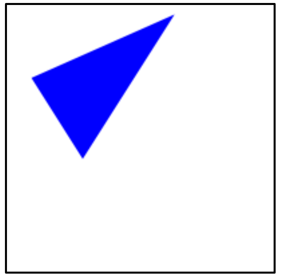
$\longrightarrow$ $a \pm 30, \text{rgb} \pm 40$

...Back to shape definition

This idea could be introduced in **shape mutation** too: it could avoid, eventually, random shape re-generation



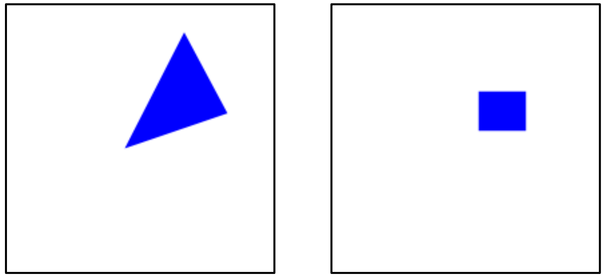
$\longrightarrow$ $Original\ Code$



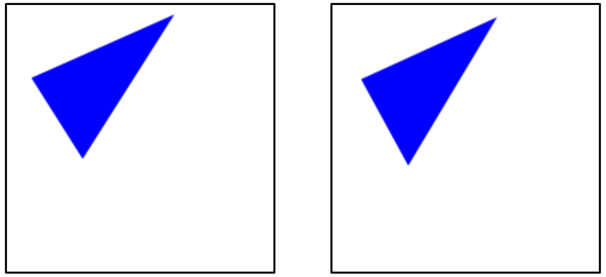
$\longrightarrow$ $x, y \pm 5$

...Back to shape definition

This idea could be introduced in **shape mutation** too: it could avoid, eventually, random shape re-generation



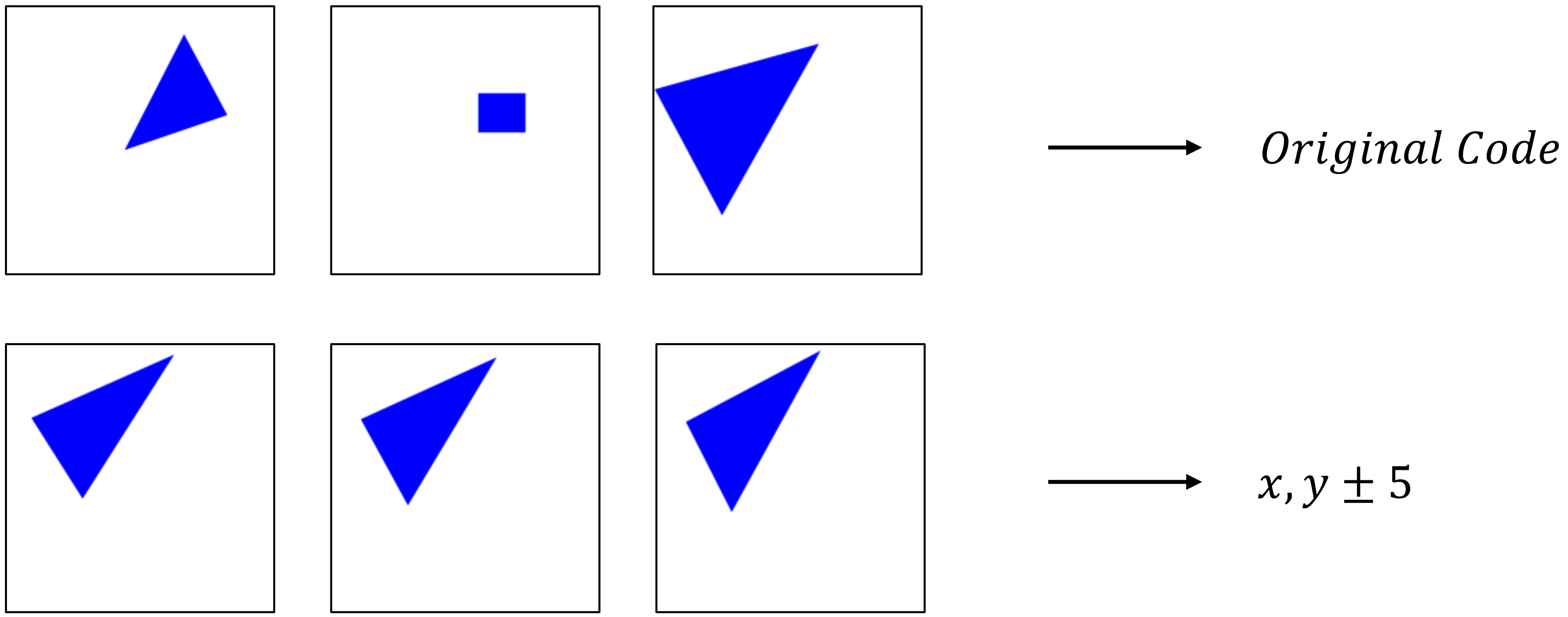
$Original\ Code$



$x, y \pm 5$

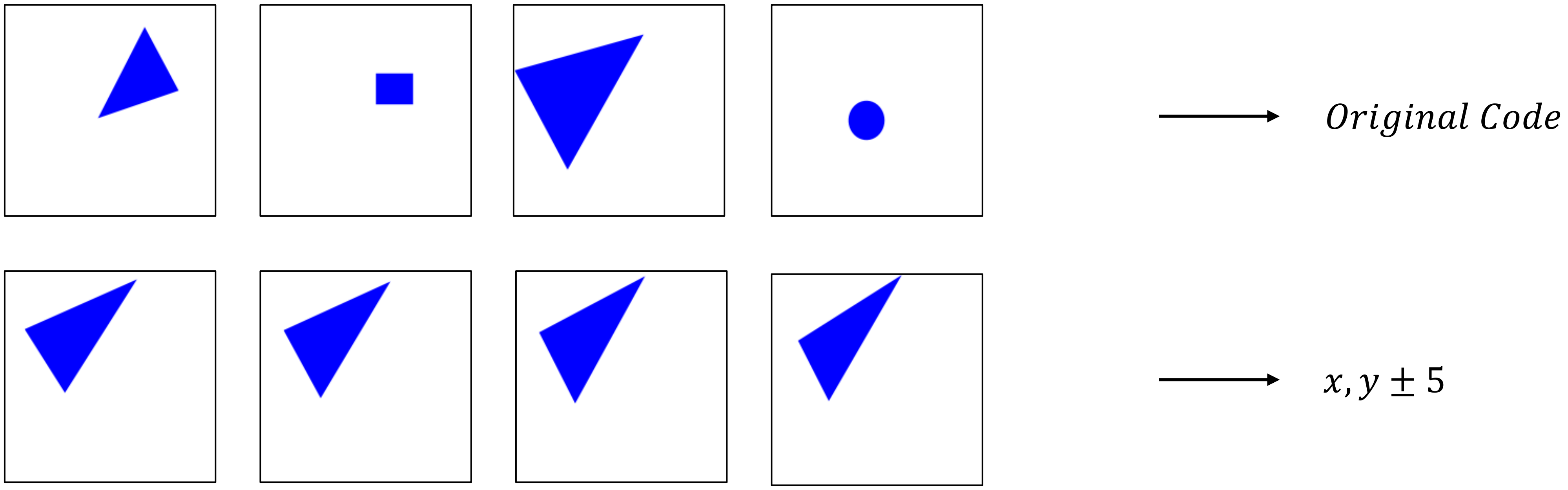
...Back to shape definition

This idea could be introduced in **shape mutation** too: it could avoid, eventually, random shape re-generation



...Back to shape definition

This idea could be introduced in **shape mutation** too: it could avoid, eventually, random shape re-generation



(modified*) Original way: performances and results

* ellipses, rectangles and triangles and shape mutation changed

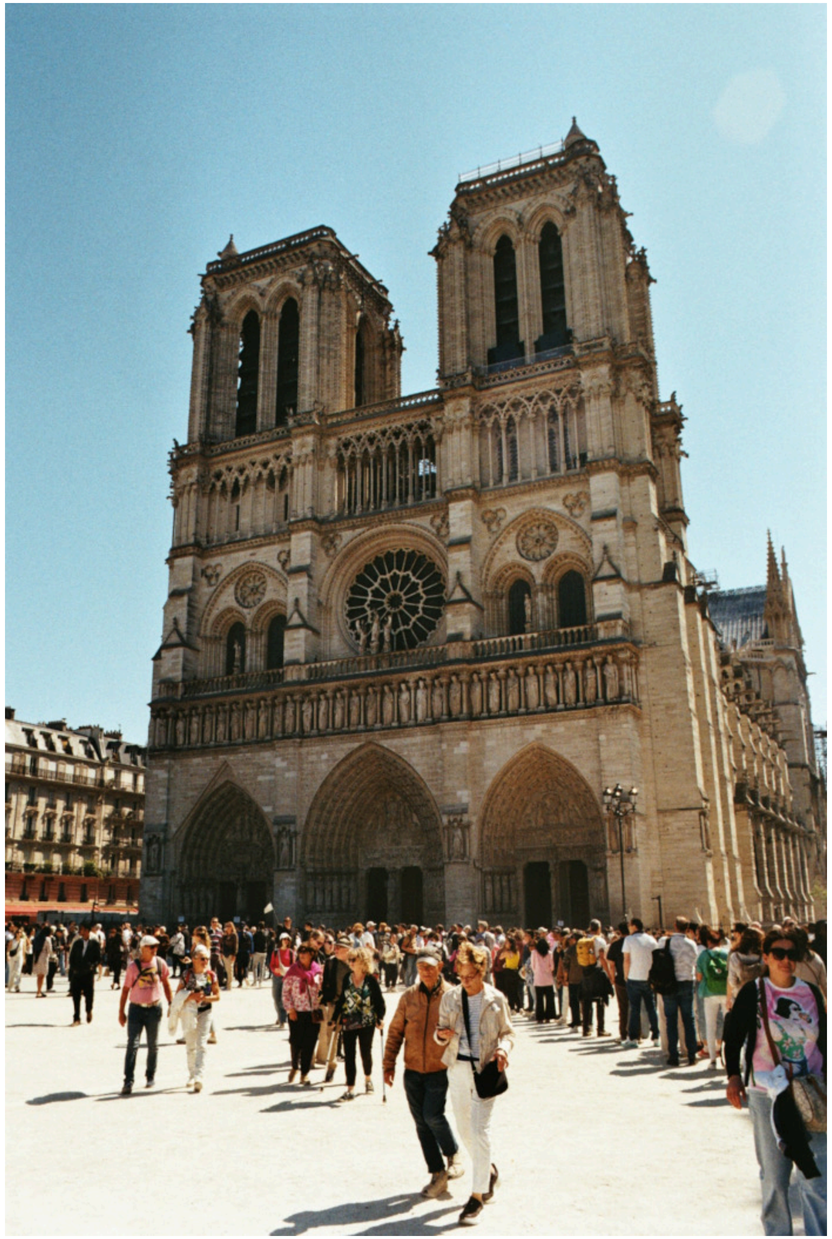
Example on Notre Dame

The example picture is a Notre Dame colored image I took this year in Paris.

The population initial parameters were

- `Image_size: 170 x 256`
- `Generation_population: 10`
- `Crossover_population: 5`

Over 30000 epochs.



The result **isn’t so good**:

- **Big shapes**
- **Computational time** is the main problem: 1h30m for 30000 epochs
- Lack in precision

Why don’t we change the approach?

How can we speed up evolution?

How can we speed up evolution?

Multiprocessing patches evolution

Fitness simplification

Patches idea introduction

The main idea is to take advantage of multi-processing.

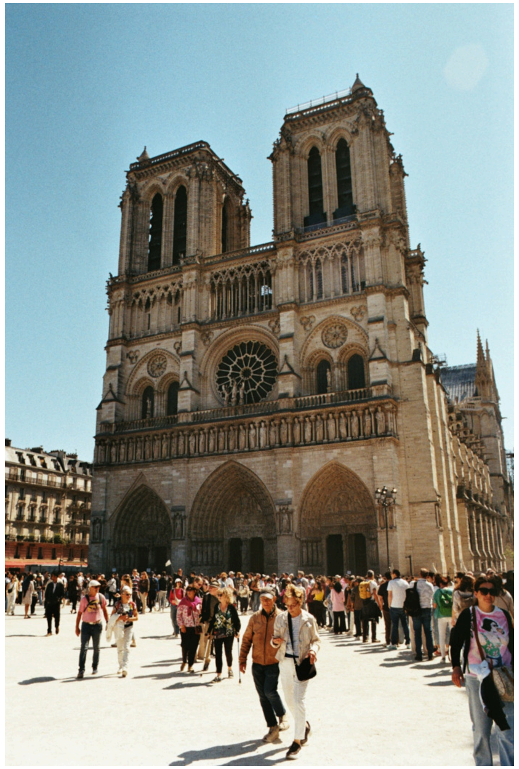
Notice this is related to the number of cores available

Also considering our **constraints**…

- The original reference should be big enough
- We cannot evolve the entire original image on different nodes

So the idea is to break and cut the image… into smaller pieces

Patches idea introduction



#processes=8

Patches idea introduction

… and evolve independently each piece.

We would have many advantages:

- Each core could handle evolution independently on a sub part of the image (patch)
- **Since shape sizes are canvas related, more precision should be present**
- We should notice a **speed increase** since fitness evaluations would be on smaller images, proportional to the number of cores
- Over same number of epochs we end up with core-number times more shapes
- **No assumptions** or reduced set of shapes and colors

+ Now we can measure shape sizes with a parameter

Fitness function revisited

+ we can define an alternative fitness which is the L1 norm:

```
def fitness(self):
        return self._fitness or self._fitness_func()
def _fitness_func(self):
    ref_img = self.reference_image.convert("RGBA")
    draw_img = np.array(self.draw().convert("RGBA"), dtype=np.int16)
    ref_img = np.array(ref_img, dtype=np.int16)
    diff = draw_img[:, :, :3] - ref_img[:, :, :3]
    fitness = np.sum(np.abs(diff))
    self._fitness = fitness
    return fitness
```

Modified way: (compared) performances and results

Time comparison

Original* Project

~**30000** Epochs

Running Time: ~ **1h 30 min**

Revisited Project

~ **40000** Epochs

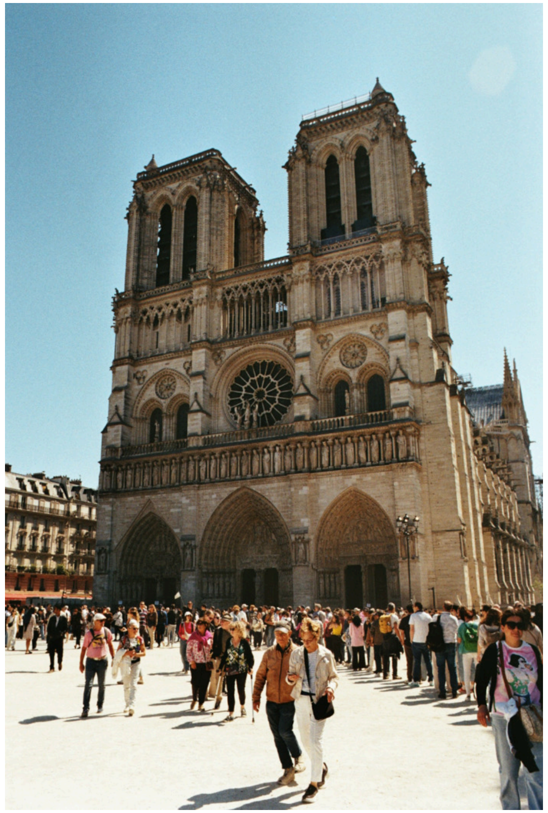
Running Time: ~ **30 min**

Same parameters

8 processes on 8 core laptop

L1 approximated fitness

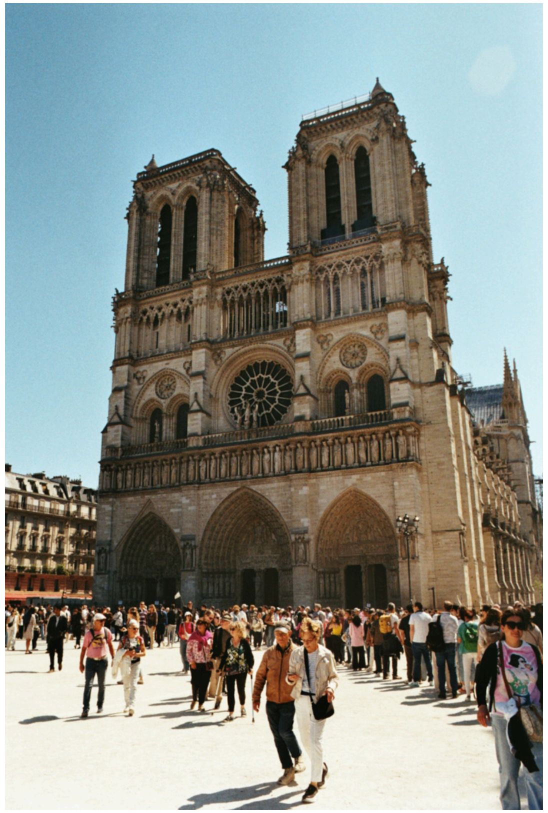
Original*



Reference

Revisited

Original*



Reference

Revisited

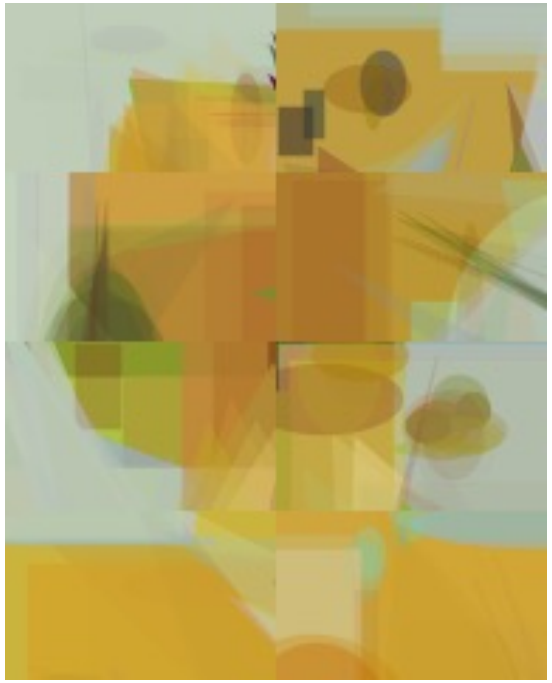
Can we do something more?

Can we do something more? → ***Shape size reduction ratio****

* w.r.t. image size

Reference

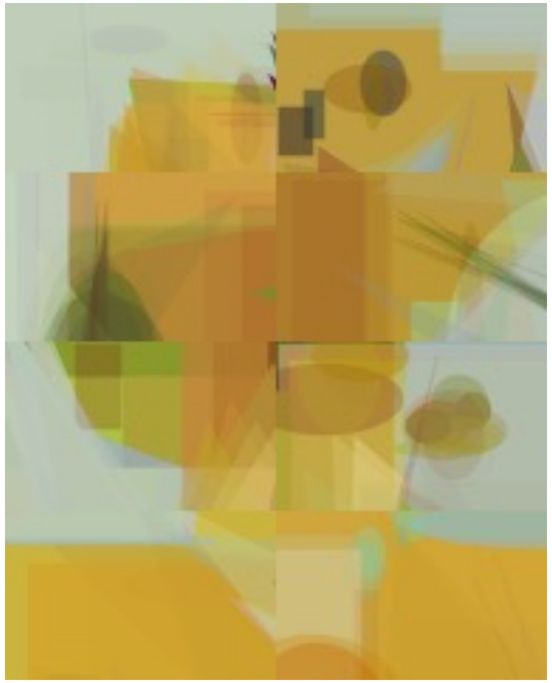
Small Shapes Evolution
1000 epochs



Big Shapes Evolution
1000 epochs

We see a great improvement with a **higher shape size reduction rate** on fixed same conditions

Small Shapes



Big Shapes

We can finally see results with:

- **Patches** and multiprocessing introduction
- More efficient **fitness**
- **Random re-generation avoidance** for shapes and colors
- Better shape random generation
- Shape size **ratio** introduction

Reference

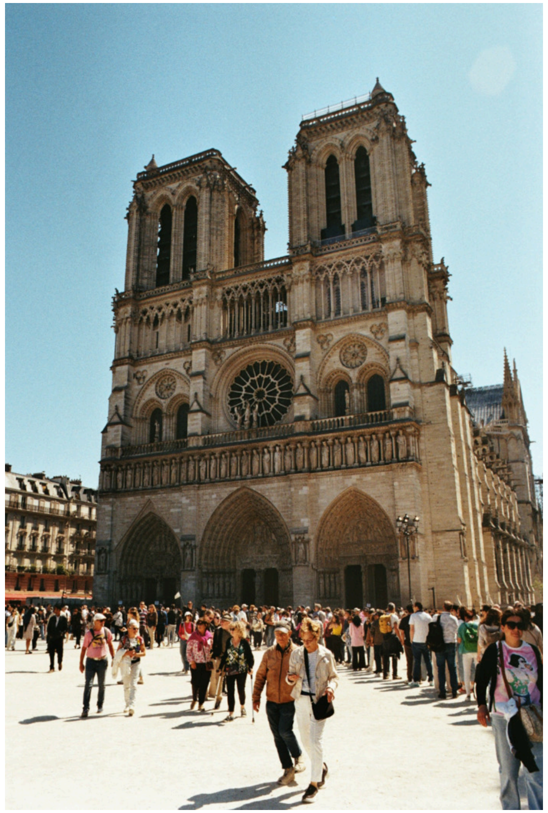
Small Shapes Evolution
50000 epochs

Reference

Small Shapes Evolution
50000 epochs

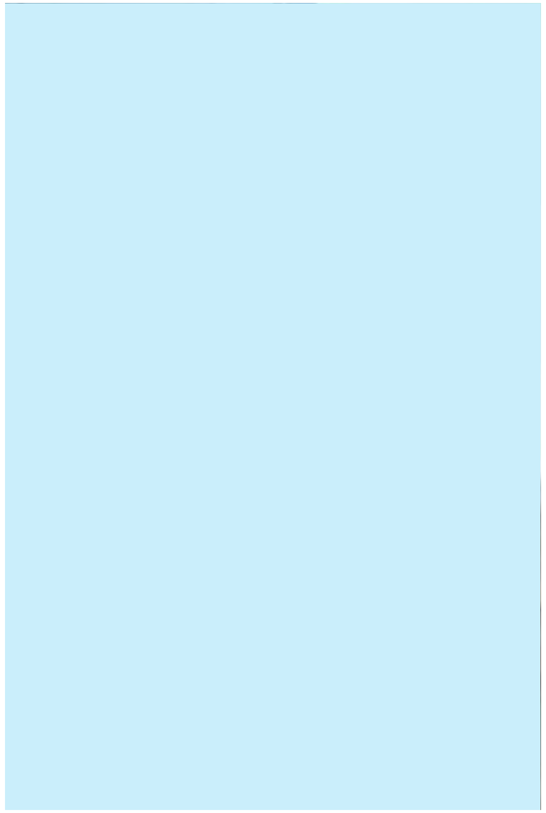
Conclusion on Notre Dame visual comparison

Original*



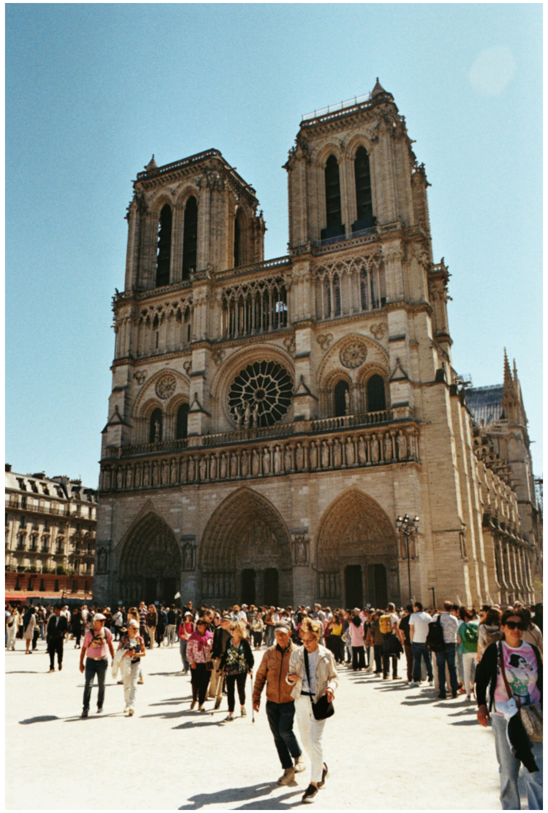
Reference

Revisited



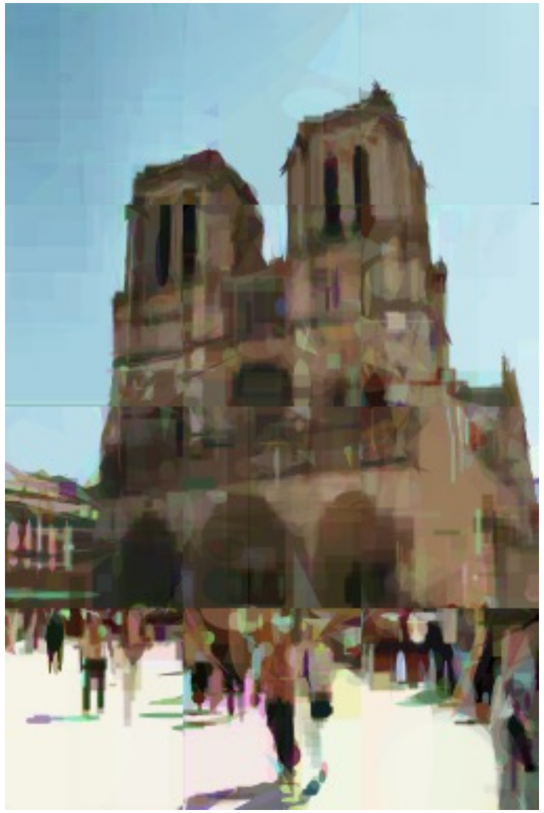
Small shape revisited

Original*



Reference

Revisited



Small shape revisited

Project References

[1] The evolution of a Smile, Peter Braden: https://github.com/peterbraden/genetic-lisa/

[2] Mona Lisa Gif Evolution: https://github.com/peterbraden/genetic-lisa/blob/master/images/lisa-anim.gif

[3] Vase with Twelve Sunflowers (Arles, August 1888), Van Gogh. Neue Pinakothek, Munich: https://commons.wikimedia.org/wiki/File:Vincent_Willem_van_Gogh_128.jpg

[4] La persistenza della memoria, Dalì: https://www.analisidellopera.it/wp-content/uploads/2018/10/Dali_La_persistenza_della_memoria-1.jpg