

# Raport z projektu malowania obrazu trójkątami

## Algorytmy Ewolucyjne

Piotr Piesiak

28 stycznia 2022

## 1 Wstęp

### 1.1 Zarys problemu

Niniejszy raport opisuje problem optymalizacji polegający na pomalowaniu określona liczbą trójkątów danego obrazu. Przez malowanie rozumie się naniesienie trójkątów z wybranymi kolorami i przezroczystością na obraz oraz ustalenie ich warstw.

### 1.2 Opis podejścia do rozwiązania

Przestrzenią przeszukiwania jest zbiór obrazów z naniesionymi  $n$  trójkątami z ustaloną kolejnością, kolorami i przezroczystością. Chromosomami są wektory zawierające dane o  $d$  trójkątach i ich kolorach. Każdy trójkąt opisany jest w postaci 3 par i jednego wektora o długości 4. Każda para  $(x, y)$  wskazuje na punkt o współrzędnych  $x, y$ , natomiast wektor o długości 4 koduje kolory i współczynnik przezroczystości  $RGBA$ . Dodatkowo kolejność trójkątów w chromosomie mówi o kolejności warstw - to znaczy, jeśli  $i < j$  to trójkąt na pozycji  $i$  będzie pod trójkątem na pozycji  $j$ . Korzystając z biblioteki numpy, można opisać chromosomy jako parę dwóch macierzy - jednej o wymiarach  $(d, 3, 2)$  opisującej pozycje trójkątów i drugiej o wymiarach  $(d, 4)$  opisującej kolory.

Po wielu próbach można dojść do wniosku, że przechowywanie w chromosomie wszystkich trójkątów - np. 100 nie daje sensownych wyników. Zastosowano więc pomysł polegający na optymalizowaniu i zatrzymaniu trójkątów w miejscu. To znaczy, że w chromosomie przechowujemy dane o  $k$  trójkątach i po określonym czasie  $t$  iteracji zatrzymujemy je i resetujemy naszą populację ustawiając nowe chromosomy z losowymi trójkątami. Takie podejście przyniosło dobre efekty.

Zastosowany algorytm jest lekką modyfikacją algorytmu pozanego na wykładzie opierającego się o strategię ewolucyjną. Poniżej przedstawiono zarys pseudokodu tego algorytmu.

```
def evolve():
    P = generate_random_population()
    population_evaluation(P, F)
    while termination_condition:
        P_c = get_children(P)
        mutate(P_c)
        replacement(P_c, P) # mu + lambda replacement
    population_evaluation()
```

Funkcja celu obliczana jest jako MSE między oryginalnym obrazem i aktualnym. Obraz przechowujemy jako macierz o wymiarach  $(height, width, 4)$  kodującą każdy piksel na płótnie. Oczywiście minimalizujemy tę funkcję.

```
def objective_function_mse(og_picture, evaluated_picture):
    x1 = np.asarray(og_picture, dtype=int)
    x2 = np.asarray(evaluated_picture, dtype=int)
    return np.sum((x1 - x2)**2) / (width * height)
```

Reprodukcja polega na wybraniu metodą ruletki rodziców z populacji i stworzeniu dzieci metodą Global Intermediere Recombination. Mutacja chromosomu polega na zmodyfikowaniu każdego punktu z ustalonym małym prawdopodobieństwem  $mut\_ppb$  o pewną losową odległość (przeskalowaną według rozmiarów obrazu). W pseudokocie celowo zastosowano pętle w celu zwiększenia czytelności - w kodzie korzysta się z szybkich operacji na obiektach numpy:

```
for triangle in chromosome:
    for coord in triangle:
        if np.random.rand() < mut_ppb:
            coord += sigma * np.random.randn(2)
```

Proces mutacji kolorów przebiega podobnie. Ograniczenie liczby osobników działa w standardowy sposób  $replacement(\mu + \lambda)$ .

### 1.3 Optymalizacja czasu

Ponieważ przestrzeń przeszukiwania jest duża, istotne jest aby ograniczyć czas działania poszczególnych funkcji. Okazuje się, że biblioteka python PIL obsługująca obrazy świetnie sprawdzi się do nanoszenia trójkątów na płótno (znacznie szybciej od nanoszenia trójkątów własną funkcją korzystającą z pętli for). Tak samo przy liczeniu funkcji celu i mutacji ograniczono do minimum liczbę pętli for na obiektach pythonowych oraz zastosowano operacje na macierzach numpy.

## 2 Eksperymenty

Z powodu długiego czasu obliczeń eksperymenty wykonywano na małej liczbie iteracji. Aby móc porównywać eksperymenty, zmieniono tylko niektóre parametry. W każdym przykładzie ilość trójkątów wynosiła 80.

### 2.1 Prawdopodobieństwo mutacji

Szybko można zauważyc, że prawdopodobieństwo mutacji nie powinno być zbyt duże - gdyż zaczynamy przeszukiwać losowo. Sprawdzono kilkukrotnie wartości prawdopodobieństw 0.1, 0.3, 0.5. Obserwacje wartości funkcji celu:

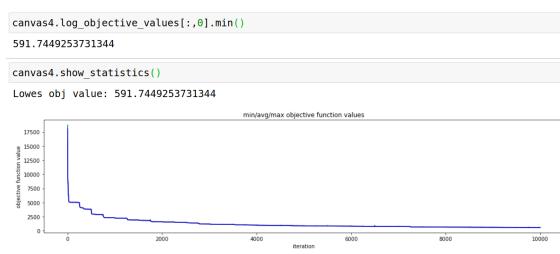
1. Algorytm z mutacją o prawdopodobieństwie 0.3 robił największe postępy ze wszystkich algorytmów na początku ewolucji (do 0.2 czasu iteracji).
2. Algorytm z prawdopodobieństwem 0.1 radził sobie najlepiej pod koniec - gdy należało dopasować mniejsze trójkąty.
3. Prawdopodobieństwo mutacji na poziomie 0.5 było zbyt duże i ten algorytm odstawał od pozostałych.

Poniżej przedstawiono wyniki eksperymentów. Celem było wykrycie efektów zmian, stąd jakości obrazów nie są bardzo dobre.



Rysunek 1: Kolejne efekty dla prawdopodobieństw: 0.1, 0.3, 0.5

Wykresy zmian wartości funkcji celu były bardzo podobne, różniły się jednak końcowymi wynikami. Dla kolejnych prawdopodobieństw 0.1, 0.3, 0.5, najlepszymi wynikami były: 626, 643, 591.



Rysunek 2: Zmiany wartości funkcji celu dla prawdopodobieństwa 0.3

## 2.2 Wielkość populacji i liczba dzieci

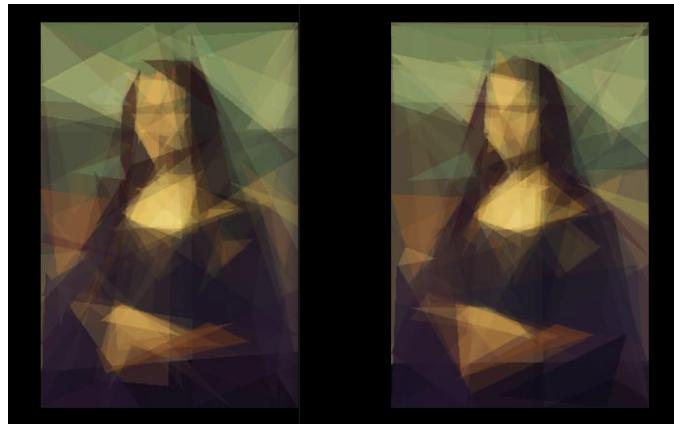
Zgodnie z oczekiwaniami większa liczba dzieci i populacji powodowała większą dokładność, ale niestety też dłuższe obliczenia.



Rysunek 3: Kolejne efekty dla wielkości populacji: 10, 30, 50

Najlepsze wartości funkcji celu osiągały obrazy z wielkością populacji 10 i 50. Najgorzej wypadł obraz z populacją rozmiaru 1. W dalszych krokach zdecydowano częściej korzystać z populacji o rozmiarze 10, gdyż daje ona zadowalające efekty przy nie tak dużym nakładzie czasu. Populacja o wielkości 50 też była kilka razy wykorzystana.

Zgodnie z oczekiwaniami im dłużej pozwolimy ewoluować trójkątom przed ich zatrzymaniem, tym dokładniejsze wyjdą obrazy. Oto wyniki dla różnych czasów ustawiania 2 trójkątów w chromosomie: Można dostrzec,



Rysunek 4: Kolejne efekty dla czasu ustawiania 2 trójkątów: 400, 500 iteracji. Uzyskano wartości funkcji celu 572 i 568

że dłuższy czas ustawiania trójkątów owocuje większą dokładnością.

## 2.3 Skalowanie

Bardzo dobrym pomysłem na przyspieszenie obliczeń okazało się skalowanie rozwiązania z mniejszego obrazu. Na początku uruchomiono ewolucję na małym rozmiarze obrazu, a następnie przeskalowano rozwiązania na obraz większy i dokonano tam ewolucję. Wynik był świetny, gdyż po połowie czasu obliczeń uzyskano niemal 2 razy lepszy wynik.



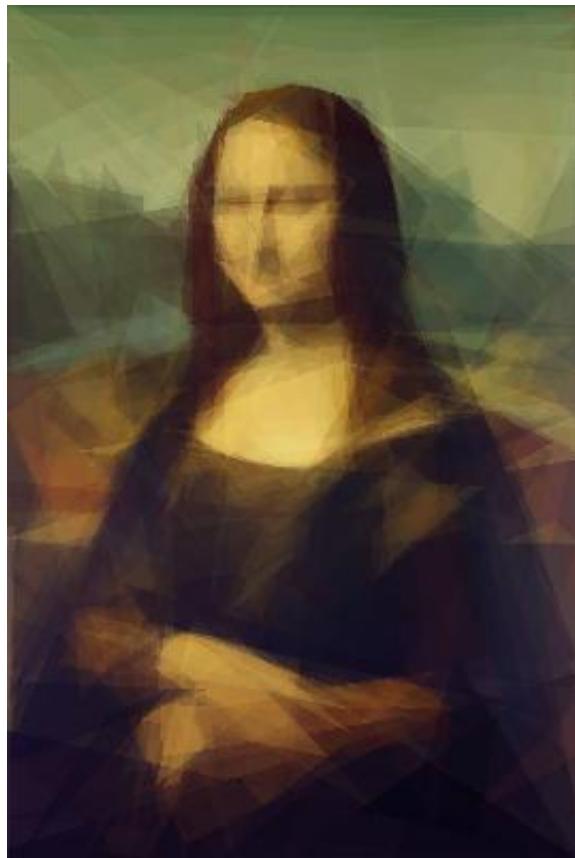
Rysunek 5: Efekty obliczeń. Po lewej bez skalowania (12h), po prawej ze skalowaniem (5h). Obraz 1908x1280pix

### 3 Podsumowanie

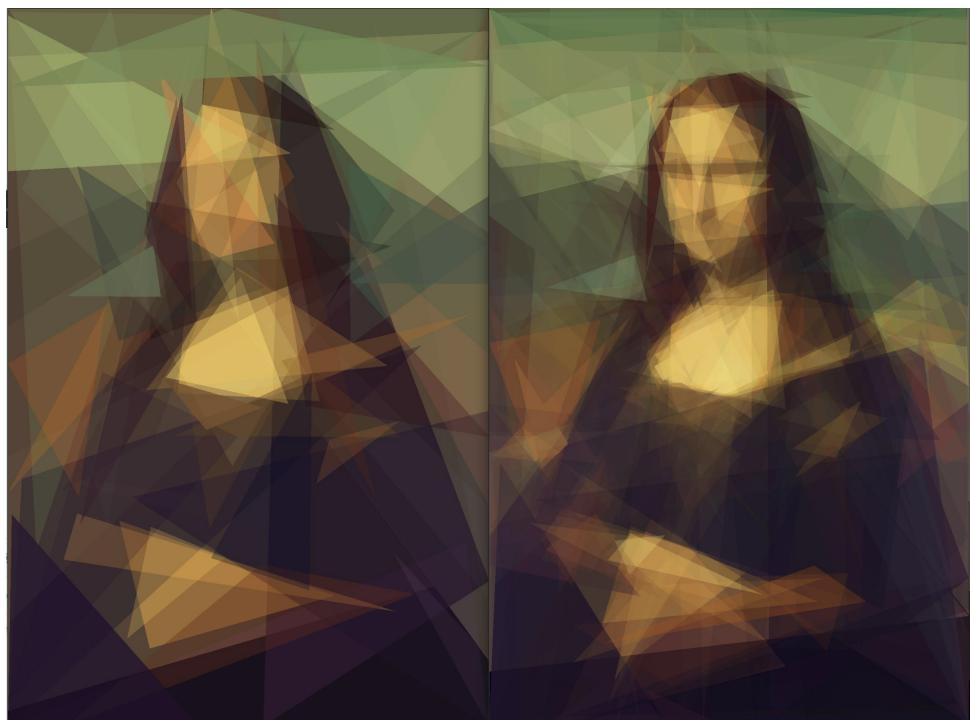
Poniżej przedstawiono wyniki dla kilku innych obrazów z ustaloną liczbą trójkątów:

#### 3.1 Wnioski i ustawienia parametrów

1. Po przeprowadzonych eksperymentach można zauważyc, że najlepsze wyniki w stosunku do czasu obliczeń uzyskuje się na populacji złożonej z około 10-20 osobników. Liczba dzieci w oczywisty sposób wpływa tylko na dokładność ustawianych trójkątów.
2. Parametry wykorzystywane w oryginalnym problemie kolorowania Mona Lisy są dosyć uniwersalne. Ze względu na dużą liczbę dzieci i osobników w populacji, nawet te bardziej szczegółowe obrazy wydają się być zadowalające.
3. W wypadku szczegółowych obrazów (takich jak np Narodziny Wenus) w celu usprawnienia obliczeń, można zmniejszyć sigmę o około 1/3 - algorytm szybciej znajdzie kontury.
4. W wypadku dużej rozdrobniości należy zwiększyć czas "ustawiania" trójkątów. Oryginalnie jest on ustalony na 250 iteracji, jednak dla Mona Lisy o rozdrobniości 1280x1908 lepiej jest go ustawić na 400 - oczywiście kosztem czasu.



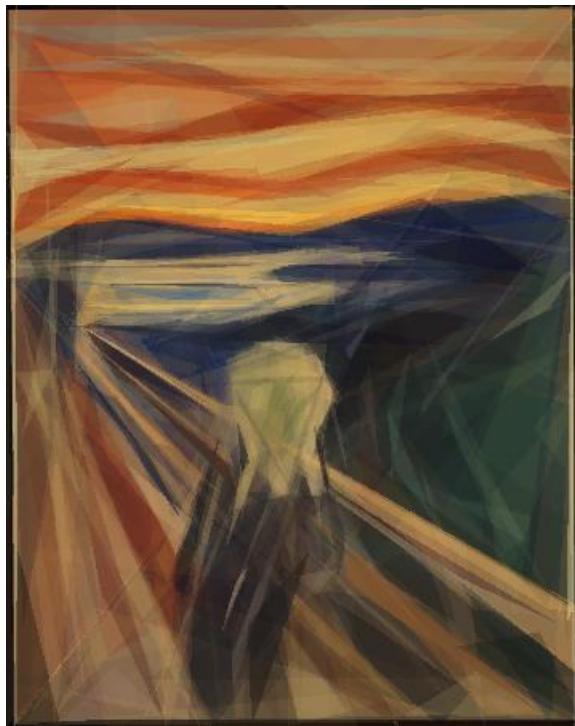
Rysunek 6: Efekt 13h obliczeń i 400 trójkątów - 278 MSE



Rysunek 7: Kolejno efekt 12h i 6h (ze skalowaniem) (większa rozdzielcość obrazu)



Rysunek 8: Bardziej skomplikowany obraz - 400 trójkątów i 11h obliczeń



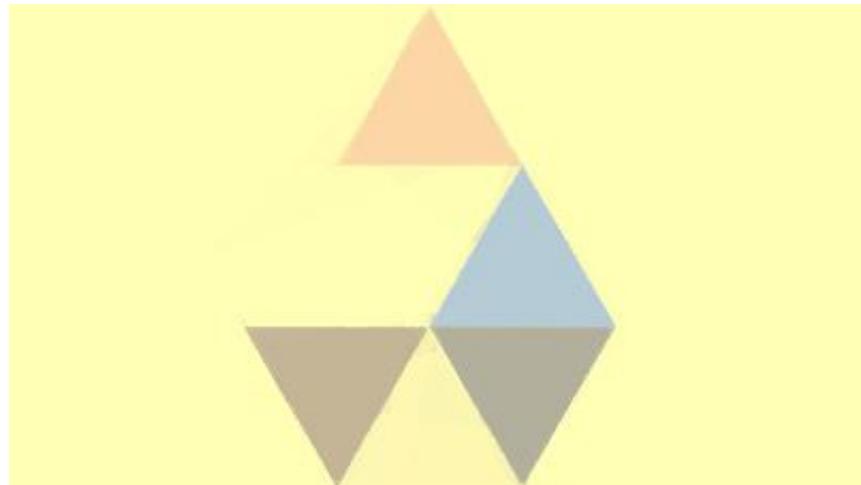
Rysunek 9: Efekt 11h obliczeń i 400 trójkątów



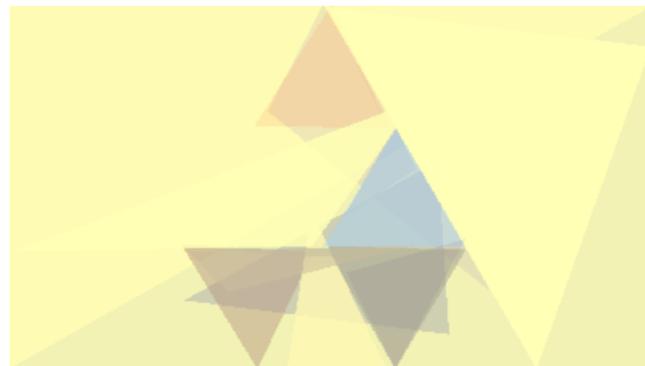
Rysunek 10: Oryginał trójkątów



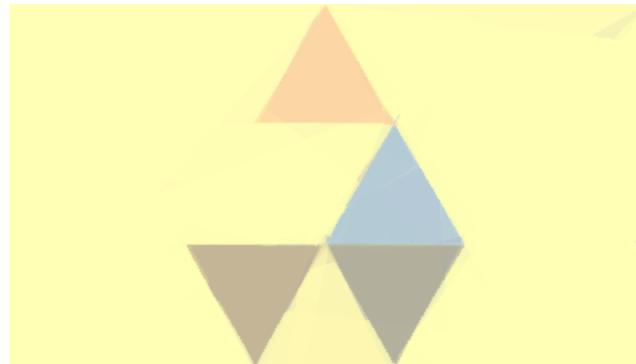
Rysunek 11: Efekt 11h obliczeń i 400 trójkątów - trudny obraz do rekonstrukcji



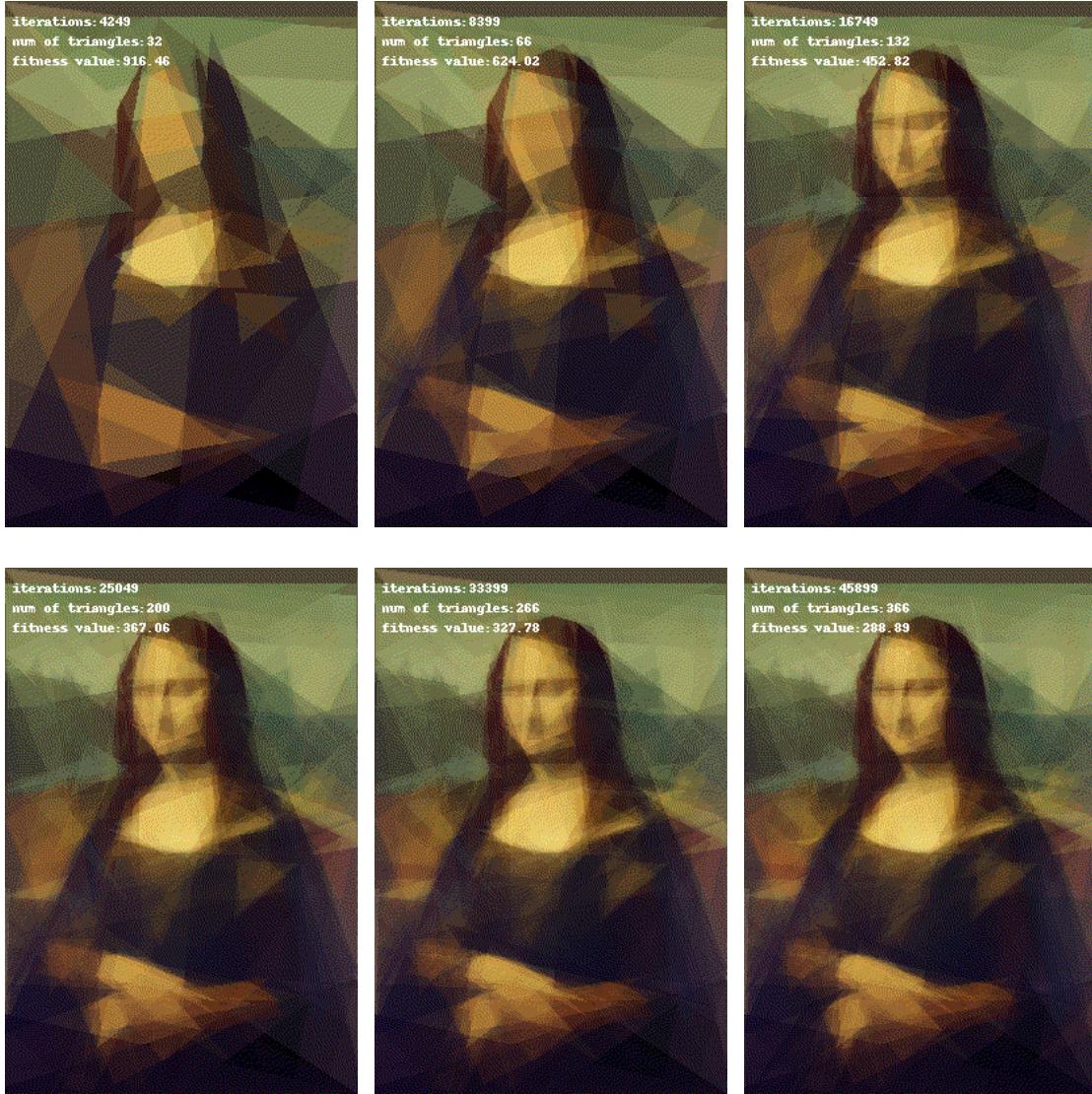
Rysunek 12: Efekt 6h obliczeń i 300 trójkątów, jednak taki stan osiąga się już po 50 trójkątach



Rysunek 13: Test po 20 trójkatach



Rysunek 14: Test po 100 trójkatach



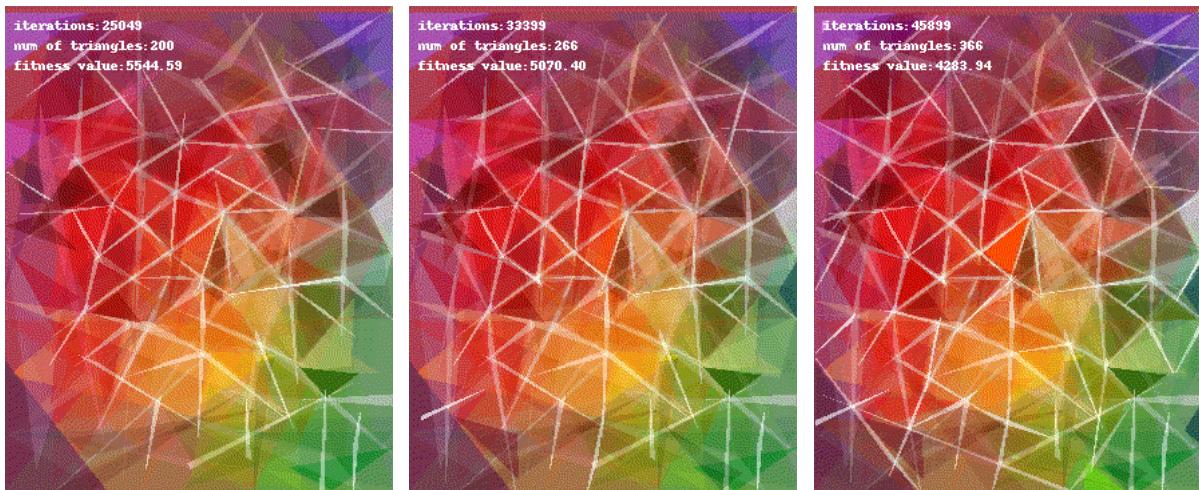
Rysunek 15: Ewolucja Mona Lisy

### 3.2 Etapy ewolucji

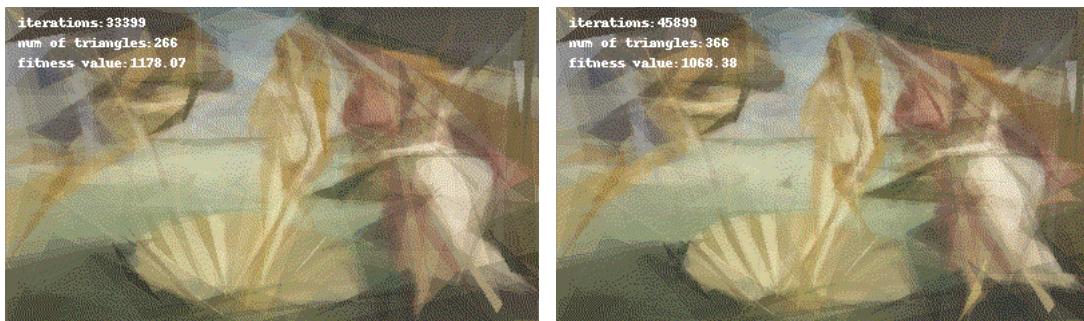
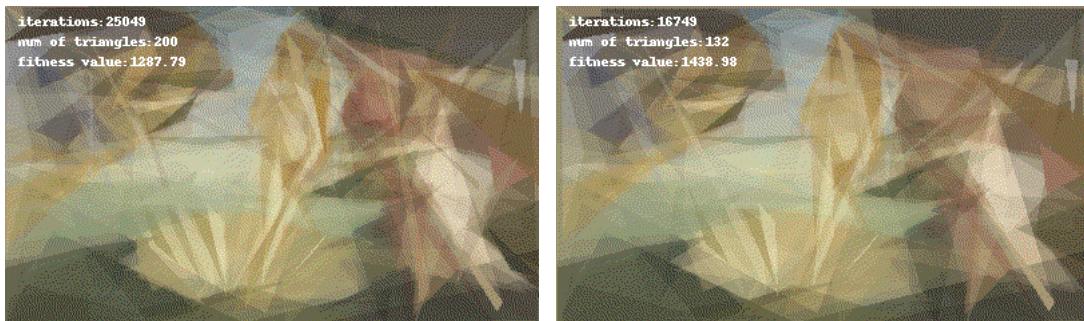
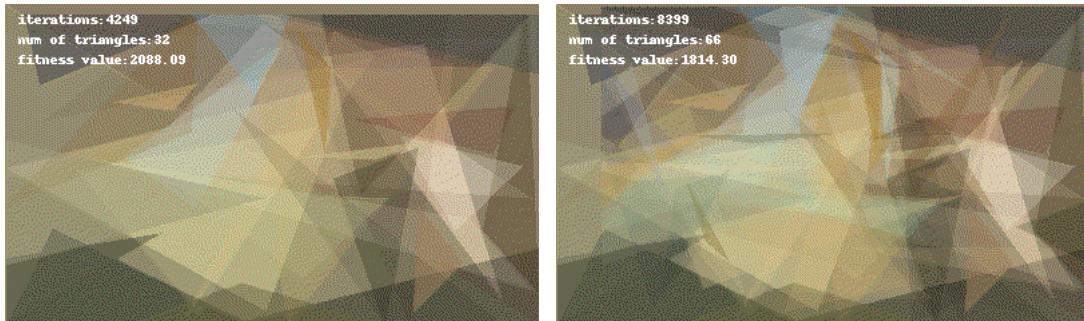


Rysunek 16: Ewolucja Krzyku





Rysunek 17: Ewolucja Trójkątów



Rysunek 18: Ewolucja Narodzin Wenus

## **Literatura**

[1] Strona wykładowa Piotra Lipińskiego.