Mateusz Jankowski

Paweł Paradowski

Tomasz Półgrabia

31.05.2015

Wektoryzacja bitmap

w zastosowaniu do rastrowych diagramów, logo firm i wzorów matematycznych

Sprawozdanie z projektu w ramach przedmiotu Podstawy Przetwarzania Obrazów.

[1 Interfejs użytkownika (GUI) 2](#_Toc420974497)

[2 Algorytm TACET 3](#_Toc420974498)

[3 Własny algorytm 4](#_Toc420974499)

[3.1 Opis algorytmu 4](#_Toc420974500)

[3.2 Zapisywanie segmentów w kolejności malejącej według z-index do pliku SVG przy użyciu biblioteki .nuget SVG 2.0 4](#_Toc420974501)

[3.3 Sposób działania algorytmu 5](#_Toc420974502)

[3.4 Sposób działania algorytmu sprawdzającego zawieranie się punktu we wielokącie 5](#_Toc420974503)

[4 Obrazy testowe 6](#_Toc420974504)

[5 Porównanie algorytmów 8](#_Toc420974505)

[5.1 Procedura testowa 8](#_Toc420974506)

[5.2 Wyniki liczbowe 9](#_Toc420974507)

[5.3 Rezultaty wektoryzacji – miniatury 12](#_Toc420974508)

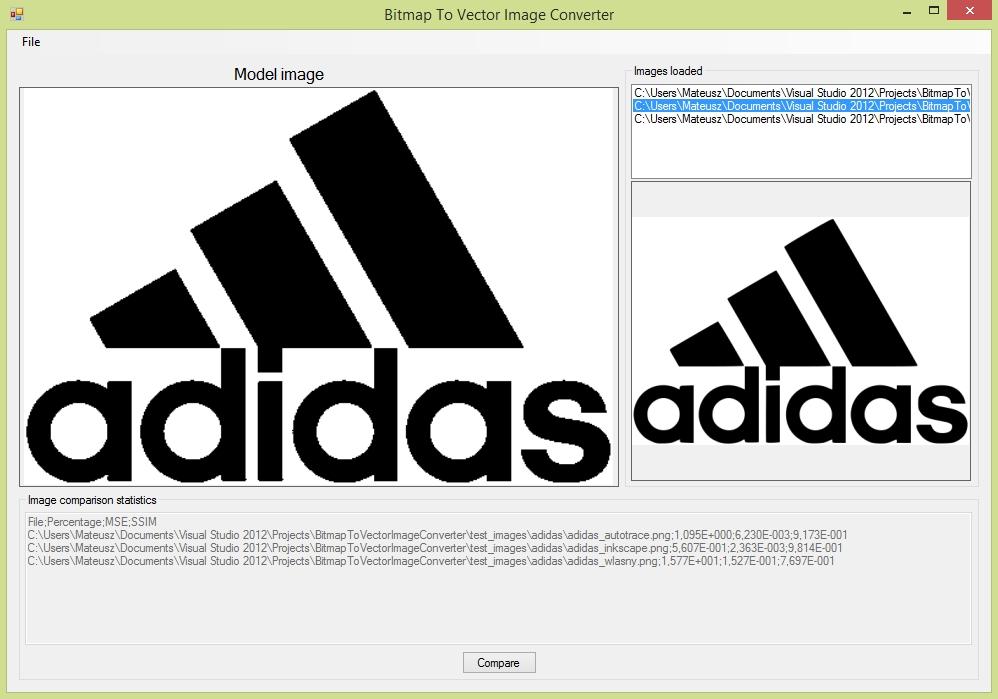
[5.4 Omówienie wyników 14](#_Toc420974509)

[6 Podział pracy 15](#_Toc420974510)

# Interfejs użytkownika (GUI)

W ramach prac nad projektem, przygotowany został graficzny interfejs użytkownika. Umożliwia on szybkie porównywanie wielu obrazów z obrazem wzorcowym. Porównywanie jest dokonywane za pomocą trzech miar: błąd MAE, błąd MSE, miara SSIM. Wyniki są prezentowane bezpośrednio na ekranie oraz zapisywane do pliku CSV. Użytkownik może obejrzeć porównywane obrazy bezpośrednio z poziomu okna aplikacji.

W pierwotnej wersji, która została odrzucona wraz z algorytmem TACET, z poziomu GUI można było dokonać wektoryzacji bitmapy.



**Rys. 1. Interfejs użytkownika**

# Algorytm TACET

Konspekt projektu zakładał implementację algorytmu TACET (Two-Arm Chain Edge Tracing), opisanego w pracy z 2008 roku: „An Efficient Algorithm for Raster-to-Vector Data Conversion”, autorstwa Junhua Teng, Fahui Wang, Yu Liu. Algorytm ten wydaje się warty uwagi, gdyż ma być ulepszeniem poprzednich pomysłów w zagadnieniu R2V dzięki wprowadzeniu koncepcji tzw. ramion. Dzięki temu przechowuje w pamięci podręcznej jedynie 2 wiersze obrazu i wykonuje tylko jeden przebieg z góry na dół.

Niestety, próba jego implementacji zakończyła się fiaskiem. Dodatkowo, nie udało nam się dotrzeć do kodu tego algorytmu np. w implementacji *Little Huskie*. Rysunki i objaśnienia, które najpierw rodziły nadzieje, że implementacja TACETu nie będzie stanowić problemu, okazały się niewystarczające. Wyjaśnienia zawarte w pracy wystarczały, żeby posiąść wyobrażenie o działaniu algorytmu, jednakże przy próbie zaaplikowania algorytmu do nawet bardzo prostych kształtów pojawiały się pytania, na które nie znaleźliśmy odpowiedzi.

Udało nam się uzyskać poprawne wyznaczanie punktów na obwodzie wielokąta tylko dla trywialnych obrazów o rozmiarze 1x1, 1x2 lub 2x1, i to przy wykorzystaniu dodatkowego przebiegu w celu oznaczenia wielokątów.

Oto najistotniejsze problemy, które spowodowały, że porzuciliśmy algorytm TACET:

1. Brak kompletnego (nawet najprostszego) przykładu działania algorytmu
2. Brak całościowego opisu działania algorytmu (od wejścia do wyjścia)
3. Skupienie tylko na wybranych szczegółach
4. Brak opisu łączenia brzegów wielokątów
5. Brak wyjaśnienia problemu tworzenia i łączenia wielokątów równolegle z tworzeniem ramion (dzięki czemu algorytm wykonuje tylko 1 przebieg)

Próby implementacji algorytmu TACET przynosiły więcej problemów, niż korzyści, wprowadzając obcy i niejasny element do procesu wektoryzacji. Rozważając proces wektoryzacji wyróżniliśmy trzy etapy:

1. Wyznaczenie obszarów spójnych wg. kolorów
2. Odpowiednie ich zareprezentowanie
3. Rozwiązanie problemu zawierania się wielokątów.

Algorytm TACET dawał jedynie nikłe szanse na rozwiązanie etapu (1), (2) tylko częściowo, ze względu na niedokładne wyjaśnienie struktur danych w pracy źródłowej, natomiast (3) jest wspomniane tylko zdawkowo jako osobny problem.

W związku z ryzykiem rosnących strat czasu i braku istotnych korzyści, podjęliśmy decyzję o zaniechaniu implementacji algorytmu TACET na rzecz własnego rozwiązania.

# Własny algorytm

## Opis algorytmu

Algorytm rozpoznawania wzorców/wektoryzujący składa się z kilku etapów:

1. Konwersja RGB na grayscale wg wzoru max(R,G,B),
2. Filtracja medianowa z zmiennym promieniem zasięgu (r = 0, brak filtracji medianowej),
3. Liczenie spójnych składowych (po kolorze),
4. Liczenie maski konturów w obrazie,
5. Liczenie obwoluty segmentów,
6. Optymalizacja kształtu obwoluty (redukcja wierzchołków) - liczenie wierzchołków,
7. Sortowanie segmentów w z-index pod względem relacji częściowego porządku – zawierania się,

## Zapisywanie segmentów w kolejności malejącej według z-index do pliku SVG przy użyciu biblioteki .nuget SVG 2.0

### Liczenie składowych/segmentów spójnych

Algorytm przechodzi sekwencyjnie po całym obrazie i dla każdego nieodwiedzonego punktu, wywołuje flood fill z indeksem nowego, utworzonego segmentu. W każdej iteracji algorytmu flood fill odwiedzony punkt jest oznaczony jako „odwiedzony”, dzięki czemu flood fill jest wywoływany tylko dla jeszcze nieoznakowanych pod względem przynależności do segmentu punktów.

### Liczenie konturu segmentu

Aby wyznaczyć maskę konturów (miejsc, w których stykają się co najmniej 2 różne segmenty), algorytm przechodzi po całym obrazie i sprawdza dla każdego piksela, czy w jego 8-spójnym otoczeniu są piksele należące do innego segmentu niż on sam. Jeśli jest to prawda, dany segment oznaczany jest jako „kontur” i na podstawie tego algorytmu tworzona jest maska konturów dla całego obrazu i segmentów.

### Liczenie obwoluty segmentu

Aby wyznaczyć obwolutę segmentu (listę punktów konturu segmentów w kolejności ciągłego, nie przerywanego przesuwania się po krawędzi w określonym kierunku) algorytm przechodzi po całym obrazie i dla każdego piksela sprawdza, czy był już on odwiedzony. Jeśli punkt nie był jeszcze odwiedzony (lub jego segment), to dla niego wykonuje flood fill, który propaguje sąsiedztwo będące konturem oraz posiadające przypisany ten sam segment co punkt startowy. Jednocześnie dla każdego punktu zapisywany jest rodzic, który propagował dany punkt (jeżeli (x,y) wypropagował punkt (qx,qy) to dla (qx,qy) (x,y) jest rodzicem).

Po skończeniu flood filla poszukiwane są wszystkie te punkty, które były odwiedzone przez algorytm i jednocześnie nie są rodzicami żadnego z propagowanych punktów w flood fillu. Przyjmuje się je, za końce ścieżek wyznaczonych przez algorytm flood fillu, odtwarza się ścieżki zaczynające się w początkowym punkcie, od którego zaczęto algorytm flood fillu i kończące w danych punktach (punktach, które nie są rodzicami innych punktów).

Ostatecznie łączy się dla każdego segmentu dwie ścieżki najdłuższe, mające jak najmniej punktów wspólnych (lewa strona obwodu i prawa strona obwodu począwszy od punktu początkowego) i w dany sposób wyznacza się kontur.

### Optymalizacja obwoluty segmentu

Do optymalizacji obwoluty segmentu służy algorytm „Ramer–Douglas–Peucker”. Jest on rekurencyjnym algorytmem do redukcji punktów w linii (obwód można potraktować jako linię zaczynającą się i kończącą się w tym samym punkcie).

1. Dla pierwszego i ostatniego punktu w liście wybierz punkt pomiędzy nimi najbardziej odległy.
2. Jeżeli odległość od linii wyznaczonej od 1. i ostatniego punktu jest większa od epsilon, rekurencyjnie liczona jest optymalna linia dla 1 i p punktu oraz p punktu i ostatniego, a następnie łączona w całość. Jeśli jest jednak mniejsza, to wynikiem algorytmu są wyłącznie punkt 1. i ostatni.

Złożoność danego algorytmu to zamortyzowane O(n log(n)), pesymistczna O(n2)

### Liczenie parametrów segmentów

W algorytmie były wykorzystywane informacje o segmentach takich jak:

* left-x, right-x, bottom-y, up-y dla prostokąta opisanego na segmencie,
* punkt ciężkości (średnia dla wszystkich pkt. po X i po Y),
* liczba pikseli w segmencie

### Zapisywanie wykrytych segmentów do plik SVG

Algorytm w celu zapisywania segmentów zapisuje listę punktów do obiektu polygon SVG. Problem pojawia się jednak, gdy jakiś wielokąt zawiera drugi wielokąt w sobie, gdyż domyślnie SVG nie oferuje możliwości operacji CSG na obiektach (union, difference, intersection). W danym celu program najpierw wyświetla te wielokąty, które zawierają w sobie inne, a w następnej kolejności te w nim zawarte (porządkuje wielokąty w z-index wg częściowego porządku relacji zawierania się wielokątów.

Aktualnie algorytm ma zaimplementowany dość powolny algorytm wykrywania zawierania się prostokątów o złożonościO(n∗m) gdzie n to liczba wierzchołków jednego wielokąta, m to liczba wierzchołków w drugim wielokącie. Aktualny algorytm był na tyle wydajny, że nie było potrzeby stosowania bardziej wydajnych algorytmów w celu przyśpieszenie jego pracy.

## Sposób działania algorytmu

1. Algorytm sprawdza czy jakakolwiek 2 krawędzie wielokąta 1. i wielokąta 2. przecinają się.
2. Jeśli TAK, zwróć wielokąt 1. nie zawiera się w pełni w wielokącie 2.
3. Jeśli NIE, sprawdź czy dowolny wierzchołek z 1. zawiera się w wielokącie 2.

## Sposób działania algorytmu sprawdzającego zawieranie się punktu we wielokącie

1. Wyznacz półprostą zaczynają się w punkcie wejściowym.
2. Wyznacz wszystkie przecięcia z krawędziami wielokąta.
3. Jeśli liczba przecięć jest nieparzysta to wierzchołek jest w środku wielokąta.

Jedyny problem do rozważenia pojawia się, gdy półprosta przecina się dokładnie w wierzchołku, ale nie przebija wielokąta.

# Obrazy testowe

Wykorzystaliśmy 14 obrazów testowych:

|  |  |
| --- | --- |
| adidas.png | adidas.png |
| bakoma.png | bakoma.png |
| bp.png | bp.png |
| cube.png | cube.png |
| dilbert.png | dilbert.png |
| godlo.png | godlo.png |
| ksztalty.png | ksztalty.png |
| landscape16.png | landscape16.png |
| nike.png | nike.png |
| prostokaty.png | prostokaty.png |
| signs.png | signs.png |
| teamliquid.png | teamliquid.png |
| text.png | text.png |
| xkcd.png | xkcd.png |

# Porównanie algorytmów

## Procedura testowa

Porównaliśmy trzy algorytmy rozwiązujące problem R2V (raster to vector):

1. Potrace autorstwa Petera Selingera (wykorzystywany przez program Inkscape, którego użyliśmy)
2. Autotrace autorstwa Martina Webera (interfejs online: autotracer.org)
3. Nasz własny algorytm (autor: Tomasz Półgrabia, patrz podział pracy)

W celu porównania jakości algorytmów, wykorzystaliśmy następującą procedurę:

1. Dokonujemy wektoryzacji obrazu
2. Renderujemy otrzymany plik .svg do formatu .png (przy pomocy programu GIMP
3. Obliczamy miary podobieństwa do oryginału

## Wyniki liczbowe

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Autotrace** | | | **Inkscape** | | | **Własny** | | |
|  | MAE | MSE | SSIM | MAE | MSE | SSIM | MAE | MSE | SSIM |
| adidas | 1.10E+00 | 6.23E-03 | 9.17E-01 | 5.61E-01 | 2.36E-03 | 9.81E-01 | 1.58E+01 | 1.53E-01 | 7.70E-01 |
| bakoma | 3.31E+00 | 2.10E-02 | 8.40E-01 | 2.03E+00 | 9.00E-03 | 9.39E-01 | 3.88E+01 | 3.78E-01 | 4.31E-01 |
| bp | 6.14E-01 | 9.66E-04 | 9.76E-01 | 5.50E-01 | 1.79E-04 | 9.97E-01 | 3.89E+01 | 3.00E-01 | 3.96E-01 |
| cube | 5.70E-01 | 1.33E-03 | 9.81E-01 | 6.69E-01 | 9.44E-04 | 9.93E-01 | 2.30E+01 | 1.83E-01 | 6.42E-01 |
| dilbert | 4.24E+00 | 2.13E-02 | 8.24E-01 | 2.87E+00 | 8.34E-03 | 9.43E-01 | SO | SO | SO |
| godlo | 1.05E+01 | 7.61E-02 | 7.24E-01 | 5.83E+00 | 2.60E-02 | 9.02E-01 | 1.80E+01 | 1.72E-01 | 3.51E-01 |
| ksztalty | 6.07E-01 | 3.85E-03 | 9.44E-01 | 2.42E-01 | 9.30E-04 | 9.92E-01 | 1.70E+00 | 1.32E-02 | 9.54E-01 |
| landscape16 | 2.49E+00 | 3.98E-03 | 8.54E-01 | 1.62E+00 | 9.24E-04 | 9.66E-01 | SO | SO | SO |
| nike | 5.45E-01 | 2.51E-03 | 9.52E-01 | 3.17E-01 | 1.21E-03 | 9.91E-01 | 3.79E+01 | 3.76E-01 | 5.89E-01 |
| prostokaty | 1.63E-01 | 6.39E-06 | 9.52E-01 | 0.00E+00 | 0.00E+00 | 1.00E+00 | 1.59E+00 | 1.13E-02 | 9.59E-01 |
| signs | 2.83E+00 | 1.13E-02 | 8.73E-01 | 1.55E+00 | 2.86E-03 | 9.71E-01 | 7.94E+00 | 4.22E-02 | 7.15E-01 |
| teamliquid | 3.84E+00 | 1.41E-02 | 8.37E-01 | 1.84E+00 | 2.28E-03 | 9.59E-01 | 2.88E+01 | 2.37E-01 | 4.46E-01 |
| text | 5.53E+00 | 4.11E-02 | 9.10E-01 | 2.63E+00 | 1.22E-02 | 9.85E-01 | SO | SO | SO |
| xkcd | 6.69E+00 | 4.86E-02 | 7.87E-01 | 3.81E+00 | 1.82E-02 | 9.11E-01 | 3.94E+01 | 3.89E-01 | 2.76E-01 |

**Tabela 1. Miary podobieństwa obrazu wyjściowego do oryginału (SO = stack overflow)**

**Wykres 1. Wielkość błędu MAE**

**Wykres 2. Wielkość błędu MSE**

**Wykres 3. Podobieństwo strukturalne do oryginału liczone miarą SSIM**

## Rezultaty wektoryzacji – miniatury

|  |  |  |  |
| --- | --- | --- | --- |
| Raster | Peter Singer potrace (Inkscape) | Martin Weber autotrace (autotracer.org) | Własny algorytm |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\adidas\adidas.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\adidas\adidas_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\adidas\adidas_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\adidas\adidas_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bakoma\bakoma.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bakoma\bakoma_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bakoma\bakoma_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bakoma\bakoma_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bp\bp.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bp\bp_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bp\bp_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\bp\bp_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\cube\cube.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\cube\cube_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\cube\cube_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\cube\cube_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\dilbert\dilbert.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\dilbert\dilbert_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\dilbert\dilbert_autotrace.png | (konwersja nieudana) |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\godlo\godlo.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\godlo\godlo_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\godlo\godlo_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\godlo\godlo_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\ksztalty\ksztalty.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\ksztalty\ksztalty_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\ksztalty\ksztalty_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\ksztalty\kszltaty_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\landscape16\landscape16.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\landscape16\landscape16_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\landscape16\landscape16_autotrace.png | (konwersja nieudana) |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\nike\nike.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\nike\nike_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\nike\nike_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\nike\nike_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\prostokaty\prostokaty.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\prostokaty\prostokaty_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\prostokaty\prostokaty_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\prostokaty\prostokaty_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\signs\signs.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\signs\signs_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\signs\signs_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\signs\signs_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\teamliquid\teamliquid.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\teamliquid\teamliquid_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\teamliquid\teamliquid_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\teamliquid\teamliquid_wlasny.png |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\text\text.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\text\text_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\text\text_autotrace.png | (konwersja nieudana) |
| D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\xkcd\xkcd.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\xkcd\xkcd_inkscape.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\xkcd\xkcd_autotrace.png | D:\Studia - Politechnika\PPO\BitmapToVectorImageConverter\test_images\xkcd\xkcd_wlasny.png |

## Omówienie wyników

Bez wątpienia najlepszym z testowanych algorytmów jest algorytm autorstwa Petera Singera – uzyskał najlepsze wyniki liczbowe dla każdego z obrazków testowych, za wyjątkiem błędu MAE dla cube.png. Znajduje to potwierdzenie wizualne, gdy spojrzymy na obrazki – nawet dla najtrudniejszych spośród nich, algorytm zdołał dokonać konwersji przy zachowaniu szczegółów i braku istotnych artefaktów – w przeciwieństwie do autotrace Martina Webera, u którego pojawia się problem białych szczelin pomiędzy wielokątami. Również jeżeli chodzi o zachowanie kształtu geometrycznego, to algorytm Petera Singera jest lepszy, niż autotrace, co widać choćby w przypadku logo firmy Adidas (niekoniecznie na miniaturze).

Algorytm własny daje zdecydowanie najsłabsze wyniki – ani wyniki liczbowe, ani miniatury nie pozostawiają tu wątpliwości. Boryka się on z artefaktami, wydajnością oraz łatwo gubi bardziej skomplikowaną geometrię. Mimo to, na uwagę zasługuje fakt, ze dla obrazków kształty.png oraz prostokąty.png uzyskał lepszą miarę podobieństwa SSIM, niż autotrace. Wnioskujemy stąd, że algorytm ma potencjał, aby konkurować z powszechnie wykorzystywanymi algorytmami R2V, choć wymagałoby to jeszcze dużo pracy. Co ważne, jest to algorytm oryginalny.

# Podział pracy

|  |  |  |
| --- | --- | --- |
| **Mateusz Jankowski** | **Paweł Paradowski** | **Tomasz Półgrabia** |
| * Tworzenie konspektu projektu * Stworzenie GUI na potrzeby algorytmu TACET oraz GUI do obliczania miar podobieństwa * Implementacja miar podobieństwa * Tworzenie sprawozdania z projektu | * Tworzenie konspektu projektu * Próba implementacji algorytmu TACET * Wybór algorytmów i obrazów do testów oraz ich przeprowadzenie * Tworzenie sprawozdania z projektu | * Tworzenie konspektu projektu * Próba implementacji algorytmu TACET * Implementacja algorytmu wektoryzacji („własny algorytm”) * Tworzenie sprawozdania z projektu |