

1 **UWAGA: wymyslic prawidlowa nazwe pracy + poprawic zlozonosci + opisac dwa algorytmy**

2 **Lord Vorotron: Finding the Best JFA Variant for the Coming** 3 **Winter**

4
5
6 MACIEJ A. CZYZEWSKI, Poznan University of Technology, Poland

7 KAMIL PIECHOWIAK, Poznan University of Technology, Poland

8 This paper studies a practical usage of machine learning (AutoML) to automate research towards discovering
9 efficient Voronoi Diagram and Distance Transform algorithms. As the baseline we used the Jump Flooding
10 Algorithm (JFA) - by finding new mutations which works best for specific data, and then ensembling them
11 into one, we create new state-of-the-art algorithm in this field named **Vorotron** with time complexity $\approx O(1)$
12 and work complexity $\approx O(N)$. The algorithm is faster and produces more accurate approximations. It could be
13 extended into 3D space in a slice-by-slice manner. We started from the assumption that JFA has potential for
14 improvement - some benefits can be observed for specific data by adding random noise and adjusting the
15 step size in JFA. This showed us that, AutoML could examine this space, and find the best possible algorithm
16 in each case. In the further part of the work, we discuss the results, compare the variants and ensemble for
17 creating the final algorithm.

18 CCS Concepts: • **Computing methodologies** → **Computer graphics; Parallel computing methodologies;**
19 **Theory of computation** → *Randomness, geometry and discrete structures; Data structures design*
20 **and analysis.**

21 Additional Key Words and Phrases: Voronoi Diagram, Distance Transform, Code Generation

22 **1 INTRODUCTION**

23
24 This paper¹ studies a practical usage of machine learning to automate research towards discovering
25 efficient Distance Transform algorithms (utilizing technique known as AutoML). Thus, by finding
26 mutations which works best for specific data, and then ensembling them into one, we create
27 new state-of-the-art algorithm in this field named Vorotron with time complexity $\approx O(1)$ and work
28 complexity $\approx O(N)$.

29 Notable contribution to the quick algorithm that makes Distance Transform (DT) using graphics
30 hardware includes [4] that creates a cone for each input (point/seed) and renders those cones
31 to obtain the Voronoi diagram as the lower envelope of these cones. [3] use planes tangent to
32 a paraboloid and thus avoid the errors caused by the tessellation of the cones. Unfortunately,
33 the drawback of this approach is the significant amount of computation and the implementation
34 complexity.

35 Jump flooding algorithm (JFA)² is an interesting way to utilize the graphics processing unit
36 to efficiently compute Voronoi diagrams and distance transforms [7]. This method is faster and
37 produces more accurate results [8], and furthermore, it could be extended into 3D space in a
38 slice-by-slice manner. This is more effective than the previous research carried out by [10], because
39 the speed of JFA is almost independent to the number of seeds [8].

40 Based on this research and findings, several efficient GPU-based algorithms which are either work
41 optimal or time optimal have been proposed including SKW [9], PBA [1], FastGPU [2], Honda's
42 algorithm [5] and WTO [6].

43
44 ¹the original title for this paper was "Lord Vorotron: Finding the Best JFA Variant for the Coming Winter"

45 ²a novel pattern of communication

46 Authors' addresses: Maciej A. Czyzewski, maciejanthonczyzewski@gmail.com, Institute of Computing Science, Poznan
47 University of Technology, Poznan, Poland; Kamil Piechowiak, kamil.cams@gmail.com, Institute of Computing Science,
48 Poznan University of Technology, Poznan, Poland.

The main question that needs to be addressed now is whether JFA has potential for improvement. We found some benefits for specific data by adding random noise and adjusting the phase size in JFA. Therefore, this shows that, AutoML could examine this unknown space, and find the best possible algorithm in each case.

For convenience, this work focus on the Voronoi diagram only - because this problem can be translated to DT [7]. The algorithm would be an approximation of the output, thus we suggest using WTO [6] for exact DT (EDT). The major contributions of this paper are thus:

- (1) Presenting new state-of-the-art variants of algorithm for Voronoi Diagram and Distance Transform:
JFAStar - (Multi-domain) single best variant; **Vorotron** - (Specific-domain) ensemble of weak variants; and
- (2) Analyzing all possible variants of JFA: comparing error and speedup relative to brute force method

2 RELATED WORK

Several efficient GPU-based algorithms which are either work optimal or time optimal have been proposed including JFA [7], SKW [9], PBA [1], FastGPU [2], Honda's algorithm [5] and WTO [6].

Reference	Algorithm	Exactness	Time	Work
[2]	FastGPU	Exact	$O(n^3/p)$	-
[1]	PBA	Exact	$O(n)$	$O(mN)$
[5]	based on SKW	Exact	$O(n)$	$O(N)$
[6]	WTO	Exact	$O(\log n)$	$O(N)$
[9]	SKW	Approximate	$O(n)$	$O(N)$
[7]	JFA	Approximate	$O(\log n)$	$O(N \log n)$
In this paper	JFAStar	Approximate	$\sim O(\log^* n)$	$\sim O(N)$
	Vorotron	Approximate	$\sim O(1)$	$\sim O(N)$

Table 1. Different GPU algorithms for computing EDT

2.1 Jump Flooding

redukcja i bridge pomiędzy intro (usunac subsection)

co to jest jump flooding? tak naprawde to nie jest algorytm do voronoi-a tylko pattern komunikacyjny w programowaniu rownoleglym - swojej pracy doktorskiej autor tej techniki podaje wiele zastosowan jednak w swoich badaniach ogranicza sie do Voronoi-a. glownym pytaniem roznych takich patternow jest ile potrzebnych jest rund/operacji komunikacji aby zagwarantowac aby dana informacja zostanie dostarczona. akurat w voronoi-u wiele komorek jest lokalna w skali calego przykladu - wiec JFA ktora gwarantuje dostarczenie informacji globalnie do kazdego punktu - wykonuje pewne niepotrzebne operacje. szybkość i zajętość pamięciowa JFA jest satysfakcjonująca, jednak proste modyfikacje pokazują że algorytm ten wykonuje się szybciej w pewnych przypadkach (i to typowych). dlatego naturalnym pytaniem powinno być w jakich oraz jakie modyfikacje wpływają na szybkość działania.

2.2 AutoML

przenieść do Proposed Method

<https://arxiv.org/pdf/1801.09373.pdf> <https://arxiv.org/pdf/1804.10120.pdf> <https://arxiv.org/pdf/1703.06353.pdf>

okay przeniosłem - dodać prace co też tak szuka algosów

3 PROPOSED METHOD

przepisać ten szkic bo język się plącze

JFA opiera się na tym że informacja jest przekazywana ??????. Przekazanie odbywa się w $\log(n)$ krokach. Więc przeprowadziliśmy krótki eksperyment aplikując losowy szum na wejściową masę. Okazało się że ilość potrzebnych kroków spadła - powstały losowe shortcuty. Co oznacza że powinny istnieć inne "mutacje" algorytmów lepsze w pewnych określonych przypadkach. Więc szukanie najszybszego algorytmu będzie następujące:

- Wymyślenie wszystkich możliwych wariantów JFA
- Mutacje i zapisanie najlepszych wersji dla danej domeny
- Ensemblacja algorytmów tak aby wybierać najlepszy wariant dla danej domeny

3.1 Domain Space

okay, zupełnie inaczej tutaj podejść, opisać jakie są przypadki i jakie są spotykane jakie domeny i dlaczego (i jak działały `gen_uniform`, `gen_polar`, `gen_grid`)

3.2 Search Space

bridge z score function gdziekolwiek to będzie obliczyć ile jest aktualnie wersji algosów np. czy jest to już 2do14 jak mamy 3xreal w wielomianie AKTUALNIE JEST około 7,200?

jakie modyfikacje, na to osobna sekcja? więc co tu napisać chyba tylko o złożoności problemu i że kod jest składany i testowany a niektóre wersje są pomijane zgodnie z działaniem `gp_minimize` (Bayesian optimization using Gaussian Processes).

w naszym wypadku zdefiniowaliśmy pewien zbiór wariantów pewnych części algorytmu (Search Space), moduł testujący dane mutacje/wariant - składa kod kernela a później go weryfikuje na naszej Domain Space.

3.3 Score Function

SCORE CZY METRIC? różnica w pikselach pomiędzy bruteforce a algorytmem - napisać o tym / też że to wszystko to ilorazy do bruteforce

dla voronoi-a interesują nas 2 parametry Error oraz Szybkość, aby wyniki były wiarygodne porównujemy je z bruteforcem (a więc będzie to iloraz). aby ocenić dane mutacje musimy przypisać jakiś Score danej wersji, więc użyliśmy wzór poniżej

$$S(x, y) = \max\{0, \sqrt{x} \cdot (100 - y^2)\}, \quad (1)$$

$$0 \leq y \leq 100, 0 < x \quad (2)$$

który kaže za zbyt wysokie error, dając zerowy wynik - składnik przy y rośnie szybciej niż x więc gdy przekroczy 100 da nam ujemny wynik - czyli 0.

3.4 Optimizer

opisać dwie osobne taktyki optymalizacji dla best single vs. ensemble

możemy napisać że korzystaliśmy z forest/gp minimize, ale też wspomnieć że aby mieć najlepszy best single to trzeba było optymalizować równocześnie całą przestrzeń (od małych do dużych, gęstych po rzadkie), a żeby mieć najlepszego Vorotrona - czyli ensembla to trzeba było dla każdej domeny z optymalizować a później jedynie zrobić balancera!!!!!!!!!!!!

trzeba to przeszukiwać tak aby nie sfiksował na zadanych parametrach - bo niechcący ocenia na początku że szum/dual jest nie fajny i później go już nie rozważa

3.5 Ensemble

można zapisać tabele dla każdej domeny (shape) / i zrobić przewidywania parametrów (słownik wariantu) - bo dla małych opłacalne są Circle6 a dla większych Circle12 i tak dalej - jak to zrobić?

patrzeć na rezultaty możemy znaleźć jaki algorytm najlepiej sprawdza się w zadanej domenie. np. widac że dla małej ilości seedów (mało gęstych przypadków, które mają małą powierzchnię) opłaca się użyć brute force. Dla kolejnych większych przypadków innych wariantów JFA. Jak wybrać algorytm? Każdy przypadek ma 'shape' oraz 'num' więc można na CPU wyselektować parę punktów albo od razu obliczyć gęstość i wybrać odpowiedni algorytm. To takich ensemblacji najlepiej sprawdzi się drzewo decyzyjne (może być boostowane).

4 VARIANTS

ZROBIC ŁADNE RYSUNEKZKI w Google Slides - eksport to pdf!

To compute the Voronoi diagram for a 2D grid of size $n \times n$ with a given set of seeds at some grid points, we are interested to propagate the content (in particular, position information) of each seed s to each grid point so that each grid point can decide which seed is its closest one.

Niektóre operacje propagacji informacji są zbędne - tylko w przypadkach rzadkich macierzy potrzeba jest $\log(n)$ kroków aby uzyskać prawidłowy wynik. Różne warianty omawiane w [8] pozwalają zredukować błąd klasycznego JFA. Nie zostały jednak omawiane przypadki gdzie poszczególne modyfikacje są używane z innymi.

Dlatego w tej pracy prezentujemy dodatkowe modyfikacje które można zastosować aby stworzyć nowe warianty. Pewne modyfikacje są oczywiste i wynikają z alternatywnego podejścia (zamiast anchoru³ kwadratowego można użyć koła), informacje można wstępnie rozpropagować losowo - w nadziei że pozwoli nam skończyć algorytm w mniejszej ilości kroków.

Aby badania były bardziej przejrzyste trzymaliśmy się pewnej konwencji nazewnictwa:

`[anchor_type][anchor_num][anchor_double]` | `[step_function]` + `[noise]`

dla przykładu `Circle11(1/3)Dual(1/4)Factor3+Noise` które można przeczytać jako:

`[anchor_type]` = *Circle*,

`[anchor_num]` = 11,

`[anchor_number_ratio]` = 1/3,

`[anchor_double]` = *True*,

`[anchor_distance_ratio]` = 1/4,

`[step_function]` = *Factor3*,

`[noise]` = *True*

³anchorem nazywamy metodę która pobiera sąsiadów do przekazania informacji

4.1 Noise

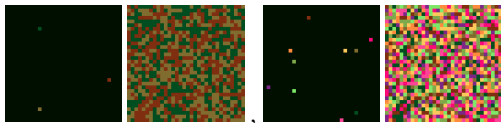


Fig. 1. Noise for 32x32

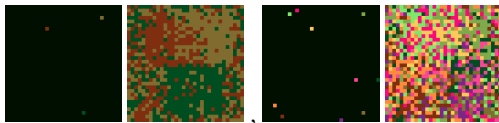


Fig. 2. Local Noise for 32x32

FIXME: figure z przykladami szumu (+local) i jak to wyglada i jak wygladalo instancja!

Zamiast zaczynac od pustej macierzy z seed-ami początkowymi można ją losowo uzupełnić szumem - tworząc przypadkowe short-cuty. Można tego dokonać osobnym kernelem który zostanie wywołany przed wykonaniem głównej części algorytmu. Interpretacja jest taka że pewne rejony która w JFA są wypełnione zerami podczas pierwszych iteracji nie podejmują żadnych decyzji. Uzupełniając szumem mogą one przypadkowo ustawić się na prawidłową wartość i propagować w kolejnej rundzie najlepszą wartość w swoim otoczeniu (zgodnie ze stepem).

dowód kamila tutaj????????????????

4.1.1 Local Noise. Można też szum uzupełniać nie losowo tylko w otoczeniu. Więc gdy w punkcie (x, y) wylosujemy losowego seed-a o wartości (x_{rand}, y_{rand}) to wyliczamy nową pozycję (x', y') która znajduje się w połowie drogi w następujący sposób: $x' = \frac{x+x_{rand}}{2}$, analogicznie dla y' . Dodatkowo jeśli (x', y') jest pusta to też uzupełniamy to pole tą informacją. Nie przejmujemy się wysiłkiem w dostępie do danych. Nadpisanie będzie losowe - a szum też.

4.2 Anchor Type

losowane punkty na okręgu???

Zamiast pobierać informacje od 8 sąsiadów o step size from grid points at $(x + i, y + j)$ where $i, j \in \{-step, 0, step\}$. Można zastosować okrąg - otwiera nam to nowe możliwości na swobodną modyfikację ilości punktów od których będziemy pobierać informacje. Naturalnie wydaje się że mała ilość punktów w anchorze spowoduje wzrost błędu, a duża ilość punktów spowoduje zmniejszenie błędu.

4.2.1 Anchor Number. Dlatego kolejnym parametrem będzie możliwość kontrolowania ilości punktów. Niestety nie rozważaliśmy wariantu kwadratów o dowolnej ilości punktów (ponieważ byłyby to wielokrotności $2 \times 2 = 4$, $3 \times 3 = 9$, $4 \times 4 = 16$, $5 \times 5 = 25$) bo i tak nie dało by się wybrać uniformly tej wartości. Dla okręgu punkty sąsiada (x_i, y_i) były liczone następująco:

$$x_i = x + step \cdot \cos\left(\frac{2\pi}{[anchor_num]} \cdot i\right), y_i = y + step \cdot \sin\left(\frac{2\pi}{[anchor_num]} \cdot i\right)$$

4.3 Anchor Double

Oprócz pojedynczego anchora, możliwe jest użycie podwójnej warstwy anchorów (czyli np. małe kołko i większe). Idea za tym stojąca to że małe kołko wewnętrzne jest dokładne (działa jak w JFA) - a wielkie zewnętrzne jest skautujące lub aby poprawić error wynikający np. z mniejszej ilości anchor_num (w sumie to podobny mechanizm jak w Lookahead - wolny/szybki)

4.3.1 Anchor Distance Ratio. Parametr mówi o stosunku długości step size od wewnętrznego anchora do zewnętrznego.

4.3.2 Anchor Number Ratio. Parametr mówi o stosunku ilości detektorów od wewnętrznego anchora do zewnętrznego.

4.4 Step Function

Gdy nasza informacja propaguje się szybciej lub jest bardziej zageszczona dlatego sąsiedzi szybciej dostają prawidłową informację - to oznacza że można skrócić ilość round wykonania algorytmu.

Step size jak i ich ilość można określić za pomocą 2 podstawowych parametrów: shape and number of points - z których później możemy określić np. średnią gęstość. Zaimplementowaliśmy 2 warianty które są uzależnione jedynie od shape: defaultowy z JFA, z JFA o podstawie 3; oraz jeden uzależniony od shape oraz od num: logstar. Jednak aby wygeneralizować problem stworzyliśmy też możliwość wygenerowania dowolnego polinomialu.

4.4.1 Special Polynomial. problem z Special - on overfituje przykłady zmieniając 5 miejsce po przecinku aby 2 zamieniało się np. w 1

Implementacja nie jest ważna - chodzi o ideę związania shape oraz num. Oraz modyfikowanie wartości, szybkości spadku, kształtu (np. pilokształnego) - jakimiś parametrami. Wada tego rozwiązania jest że trzeba optymalizować tę funkcję na całej dziedzinie (małej, dużej, gęstej, zadkiej) - bo inaczej z overfituje ona ilość kroków i wielkość kroku pod rozmiar.

```
def mod_step_function_special(shape, num=None, config=None):
    # [EXAMPLE]
    # Special(1.51/0.92/0.92/1.08/0.42)
    # ----- A -- B -- C -- D -- X ---

    A = config["A"] # <1, 2>
    B = config["B"] # <0, 1>
    C = config["C"] # <0, 1>
    D = config["D"] # <1, 2>
    X = config["X"] # <0.2, 1>

    q = num / (shape[0] * shape[1])
    qm = ((shape[0] + shape[1]) / 2) * q**(1 / 2)
    S = B * qm + (1 - B) * (max(shape) / 2)
    St = math.log2(S)

    steps = []
    for i in range(1, int(X * St * 2), 1):
        f = round(1 / (D**(i**A) + i % max(1, int(C * St))), 4)
        ffm = int(f * S)
        if ffm >= 1:
            steps.append(ffm)
    if len(steps) == 0:
        return [1]

    return steps
```

5 RESULTS

przenieść legendę? JAKO OSOBNY PDF? i podać w tej sekcji - tak się nie robi ale było by ok i czytelnie + więcej miejsca na wykresy a przypadków będzie więcej czy wykres loss oraz score dla przypadków powinny być nałożone? albo połączone subfigurem tak aby osie były sync. i dało się porównać

performance plot⁴

⁴wykres został zrobiony poprzez posortowanie score'ów - dzięki temu widac różnice w przyroście i łatwo dostrzec który algorytm ma najwyższy score lub jaka ma charakterystykę (np. jest bardzo skuteczny dla wąskiej grupy przykładów)

5.1 Multi-domain Algorithm (JFAStar)

- **shapes:** {32x32, 64x64, 96x96, 128x128, 256x256, 320x320, 384x384, 448x448, 512x512, 768x768, 1024x1024, 1536x1536}
- **cases:**
 - gen_uniform: seeds=1,
 - gen_uniform: seeds=2,
 - gen_uniform: density=0.0001,
 - gen_uniform: density=0.001,
 - gen_uniform: density=0.01,
 - gen_uniform: density=0.02,
 - gen_uniform: density=0.03,
 - gen_uniform: density=0.04,
 - gen_uniform: density=0.05,
 - gen_uniform: density=0.1,

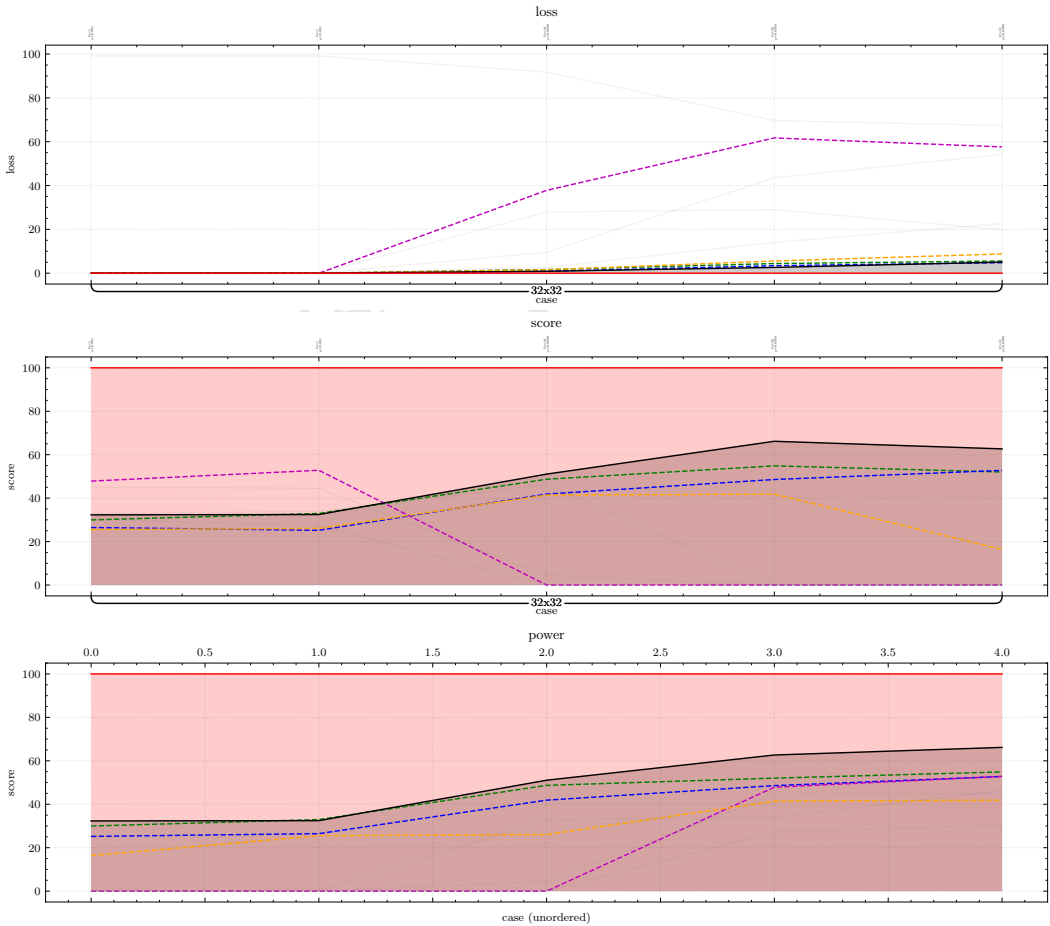


Fig. 3. bla bla bla

Algorithm	$\rho=0.001$	$\rho=0.0195$	$\rho=0.0498$	Avg. score
Square Factor3	31.5	51.8	52	44
Circle10(3/4) Special(1.68/0.45/0.98/1.12/0.81)	25.8	45.2	52.8	40
Square Special(1.0/1.0/0.0/1.0/0.2)	50.3	0	0	31
Circle11(1/3)Dual(3/4) Default	25.8	41.6	16.3	31
Circle8(3/4)Dual Factor3	33.4	23.2	0	29
Circle4(3/4)Dual(3/4) Star	44.8	2.3	0	28
Square Factor3+LNoise	28.8	0	0	18
SquareDual(2/3) Special(1.51/0.92/0.92/1.08/0.42)	0	0	0	0

Table 2. domain: 32x32, 64x64, 128x128, 256x256

5.2 Specific-domain Algorithm

5.2.1 32x32, 64x64, 96x96, 128x128.

5.2.2 256x256, 320x320, 384x384, 448x448.

5.2.3 512x512, 768x768, 1024x1024, 1536x1536.

5.2.4 Low Density.

5.2.5 High Density.

5.3 Ensemble Across Domains (Vorotron)

bla bla

5.4 Objectives

naprawic generowanie tego wykresu napisac co nie moze byc uzyte z czym?
czyli co ma wplyw na co (w sumie to najwazniejsze mialo byc w pracy)

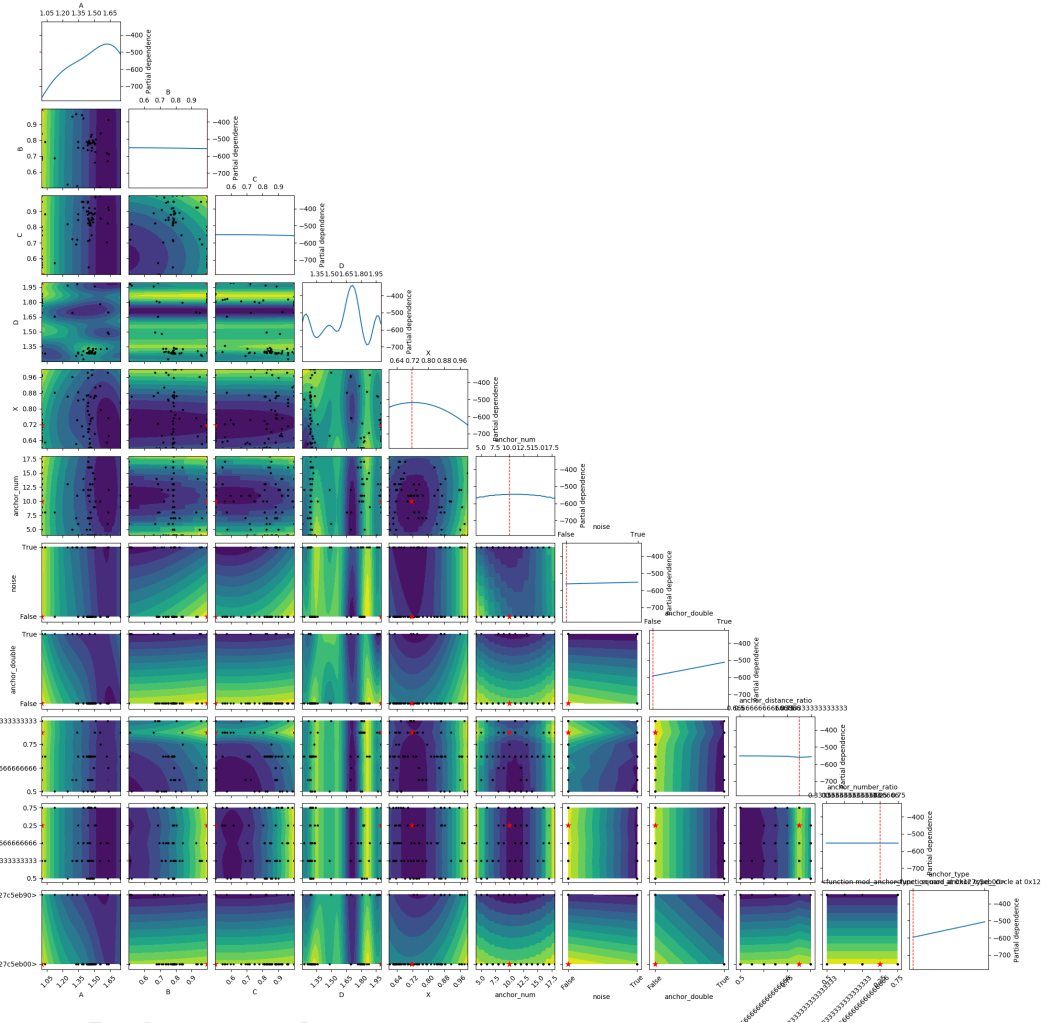


Fig. 4. bla bla bla

6 PRACTICAL USAGE

polaczyc z Conclusions

Jest wiele projektow ktore potrzebuje DT lub voronoi-a. Jedyne dwa praktyczne przyklady z tej pracy to SOTA dla JFA - czyli JFAstar, oraz praktyczny Ensemble (uwzgledniajacy np. bruteforce dla malych instancji).

7 CONCLUSIONS

This paper presents the GPU's effective, almost constant, algorithm for calculating the Euclidean distance transform (DT) approximation for 2D and higher dimensional images. As mentioned in [1], it remains challenging to balance the workload in such an approach. Vorotron does not explicitly solve this issue but, by constructing an alternative solution utilizing random shortcuts and parameter estimation, it makes it a reasonable approximation. In practice, such a constant time

algorithm is useful in many interactive applications, such as tessellations, rendering, and image processing, involving [7].

8 ACKNOWLEDGEMENTS

Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu!
Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu!
Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu!
Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu!
Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu!
Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu! Dziekuje swojemu psu!

REFERENCES

- [1] Thanh-Tung Cao, Ke Tang, Anis Mohamed, and Tiow-Seng Tan. 2010. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. 83–90.
- [2] Francisco de Assis Zampieri and Leonardo Filipe. 2017. A fast CUDA-based implementation for the Euclidean distance transform. In *2017 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 815–818.
- [3] Ian Fischer and Craig Gotsman. 2006. Fast approximation of high-order Voronoi diagrams and distance transforms on the GPU. *Journal of Graphics Tools* 11, 4 (2006), 39–60.
- [4] Kenneth E Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 277–286.
- [5] Takumi Honda, Shinnosuke Yamamoto, Hiroaki Honda, Koji Nakano, and Yasuaki Ito. 2017. Simple and fast parallel algorithms for the Voronoi map and the Euclidean distance map, with GPU implementations. In *2017 46th International Conference on Parallel Processing (ICPP)*. IEEE, 362–371.
- [6] Manduhu Manduhu and Mark W Jones. 2019. A work efficient parallel Algorithm for exact Euclidean distance transform. *IEEE Transactions on Image Processing* 28, 11 (2019), 5322–5335.
- [7] Guodong Rong and Tiow-Seng Tan. 2006. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 109–116.
- [8] Guodong Rong and Tiow-Seng Tan. 2007. Variants of jump flooding algorithm for computing discrete Voronoi diagrams. In *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE, 176–181.
- [9] Jens Schneider, Martin Kraus, and Rüdiger Westermann. 2009. GPU-based real-time discrete Euclidean distance transforms with precise error bounds.. In *VISAPP (1)*. 435–442.
- [10] Avneesh Sud, Naga Govindaraju, Russell Gayle, and Dinesh Manocha. 2006. Interactive 3D distance field computation using linear factorization. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 117–124.