LORD VOROTRON: FINDING THE BEST JFA VARIANT FOR THE COMING WINTER

DRAFT

Maciej A. Czyzewski
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznan, Poland
maciejanthonyczyzewski@gmail.com

Kamil Piechowiak
Institute of Computing Science
Poznan University of Technology
Piotrowo 2, 60-965 Poznan, Poland
kamil.cams@gmail.com

September 15, 2020

UWAGA! Przenioslem fragmenty ze starego szkicu, teraz jednak czuje ze powinnien byc to takie ogolne podsumowanie wszystkich mozliwych wariantow JFA - i w jakich przypadkach sie sprawdzaja. A tak przyokazji nasza wersja z szumem+trikami ktora dobrze dziala, no i dodatek taki ze mozna teraz zrobic sobie ensembla (a nie jako glowny cel tej pracy). Dlatego wszystko co ponizej to praktycznie random/bardzo mocny szkic. Wykresy to wizualizacja smaku.

ABSTRACT

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

CEL: omawiamy dwa algorytmy? jeden nastepca JFA - **JFAStar** oraz ensembla po naszym score fn. - **Lord Vorotron**???????

Keywords aaaaaaaaaaaaaaaaaa · bbbbbbbbbbbbbbb · ccccccccccccccccccc

1 Introduction

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

Notable contribution to the quick algorithm that makes Distance Transform (DT) using graphics hardware includes Hoff III et al. [1] that creates a cone for each input (point/seed) and renders those cones to obtain the Voronoi diagram as the lower envelope of these cones. [2] use planes tangent to a paraboloid and thus avoid the errors caused by the tessellation of the cones. Unfortunately, the drawback of this approach is the significant amount of computation and the implementation complexity.

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

Jump flooding algorithm (JFA)[2] is an interesting way to utilize the graphics processing unit to efficiently compute Voronoi diagrams and distance transforms [3]. This method is faster and produces more accurate results [5], and furthermore, it could be extended into 3D space in a slice-by-slice manner. This is more effective than the previous research carried out by Sud et al. [4], because the speed of JFA is almost independent to the number of seeds [5].

Based on this research and findings, several efficient GPU-based algorithms which are either work optimal or time optimal have been proposed including SKW [6], PBA [7], FastGPU [8], Honda's algorithm [9] and WTO [10].

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 [3]. The algorithm would be an approximation of the output, thus we suggest using WTO [10] 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** - single best variant; **Lord Vorotron** - ensemble of weak variants; and
2. Analyzing all possible variants of JFA: comparing error and speedup relative to bruteforce method

2 Related Work

Several efficient GPU-based algorithms which are either work optimal or time optimal have been proposed including JFA [3], SKW [6], PBA [7], FastGPU [8], Honda's algorithm [9] and WTO [10].

Reference	Algorithm	Exactness	Time	Work
Assis Zampirolli et al. [8]	FastGPU	Exact	$O(n^3/p)$	-
Cao et al. [7]	PBA	Exact	$O(n)$	$O(mN)$
Honda et al. [9]	based on SKW	Exact	$O(n)$	$O(N)$
Manduhu et al. [10]	WTO	**Exact**	$\boldsymbol{O(\log n)}$	$O(N)$
Schneider et al. [6]	SKW	Approximate	$O(n)$	$O(N)$
Rong et al. [3]	JFA	Approximate	$O(\log n)$	$O(N \log n)$
In this paper	Lord Vorotron	**Approximate**	$\boldsymbol{\sim O(1)}$	$\sim O(N)$

Table 1: Different GPU algorithms for computing EDT

2.1 Jump Flooding

redukcja i bridge pomiedzy 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. szybkosc i zajetosc pamieciowa JFA jest satysfakcujaca, jednak proste modyfikacje pokazuja ze algorytm ten wykonuje sie szybciej w pewnych przypadkach (i to typowych). dlatego naturalnym pytanie powinno byc w jakich oraz jakie modyfikacje wplywaja na szybkosc dzialania.

2.2 AutoML

przeniesc do Proposed Method

okay przenioslem - dodac prace co tez tak szuka algosow

[2] a novel pattern of communication

3 Proposed Method

przepisac ten szkic bo jezyk sie placzy

JFA opiera sie na tym ze infomacja jest przekazywana ??????. Przekazanie odbywa sie w log(n) krokach. Wiec przeprowadzilismy krotki eksperyment applyujac losowy szum na wejsciowa masce. Okazalosie sie ze ilosc potrzebnych krokow spadla - powstaly losowe shortcuty. Co oznacza ze powinny istniec inne "mutacje" algorytmow lepsze w pewnych okreslonych przypadkach. Wiec szukanie najszybszego algorytmu bedzie nastepujace:

- Wymyslenie wszystkich mozliwych wariantow JFA
- Mutacje i zapisanie najlepszych wersji dla danej domeny
- Ensemblacja algorytmow tak aby wybierac najlepszy variant dla danej domeny

3.1 Domain Space

jakie domeny i dlaczego (i jak dzialaly gen_uniform, gen_polar, gen_grid)

- **shapes**: {(64, 64), (128, 128), (256, 256), (512, 512), (768, 768)}
- **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,

3.2 Search Space

bridge z score function gdziekolwiek to bedzie obliczyc ile jest aktulanie wersji algosow np. czy jest to juz 2do14 jak mamy 3xreal w wielomianie AKTUALNIE JEST okolo 7,200?

jakie modyfikacje, na to osobna sekcja? wiec co tu napisac chyba tylko o zlozonosci problemu i ze kod jest skladany i testowany a niektore wersje sa pomijane zgodnie z dzialaniem gp_minimize (Bayesian optimization using Gaussian Processes).

w naszym wypadku zdefiniowalismy pewien zbior variantow pewnych czesci algorytmu (Search Space), modul testujacy dana mutacje/wariant - sklada kod kernela a pozniej go weryfikuje na naszej Domain Space.

3.3 Score Function

SCORE CZY METRIC? roznica w pikselach pomiedzy bruteforce a algorytmem - napisac o tym / tez ze to wszystko to ilorazy do bruteforce

dla voronoi-a interesuja nas 2 parametry Error oraz Szybkosc, aby wyniki byly wiarygodne porownujemy je z bruteforcem (a wiec bedzie to iloraz). aby ocenic dana mutacje musimy przypisac jakis Score danej wersji, wiec uzylismy wzor ponizej

$$S(x, y) = max\{0, x \cdot (100 - y^{1.5})\}, \tag{1}$$
$$0 \leq y \leq 100, 0 < x \tag{2}$$

ktory kaze za zbyt wysokie errory, dajac zerowy wynik - skladnik przy y rosnie szybciej niz x wiec gdy przekroczy 100 da nam ujemny wynik - czyli 0.

3.4 Optimizer

opisac dwie osobne taktyki optymalizacji dla best single vs. ensemble

mozemy napisac ze korzystalismy z forest/gp minimize, ale tez wspomniec ze aby miec najlepszy best single to trzeba bylo optymalizowac rownoczesnie cala przestrzen (od malych do duzych, gestych po rzadkie), a zeby miec najlepszego Vorotrona - czyli ensembla to trzeba bylo dla kazdej domeny z optymalizowac a pozniej jedynie zrobic balancera!!!!!!!!!!!!

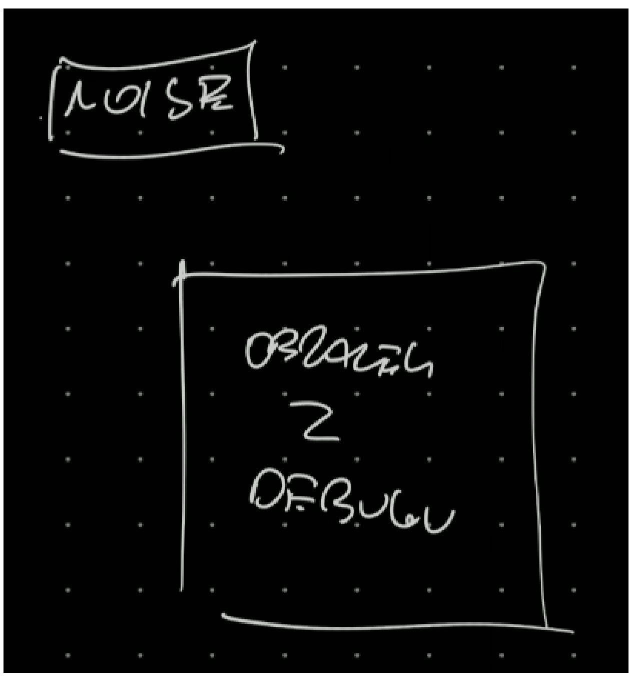
3.5 Ensemble

patrzac na rezultaty mozemy znalesc jaki algorytm najlepiej sprawdza sie w zadanej domenie. np. widac ze dla malej ilosci seedow (malo gestych przypadkow, ktore maja mala powierzchnie) oplaca sie uzyc bruteforce. Dla kolejnych wiekszych przypadkow innych wariantow JFA. Jak wybrac algorytm? Kazdy przypadek ma 'shape' oraz 'num' wiec mozna na CPU wysemplowac pare punktow albo odrazu obliczyc gestosc i wybrac odpowiedni algorytm. To takich ensemblacji najlepiej sprawdzi sie drzewo decyzyjne (moze byc boostowane).

4 Variants

ZROBIC LADNE RYSUNECZKI w Google Slides - eksport to pdf! wyjasnic slownictwo? np. co okreslamy przez anchor i wyjasnic jak powstaly nazwy? np. Circle11DuallFactor3+Noise

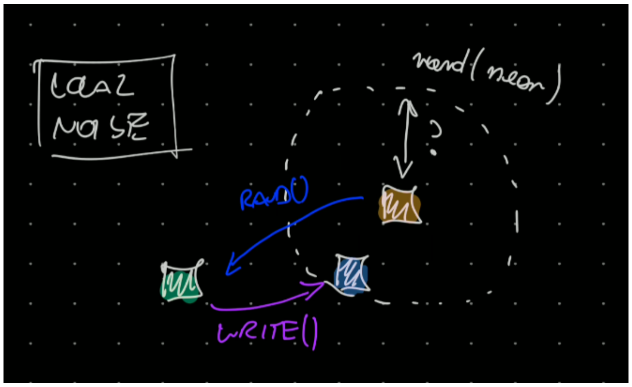
4.1 Noise



w poczatkowej wersji dodanie szumu pozwolilo zmienic zlozonosc - dodatkowo mamy na to dowod kamila

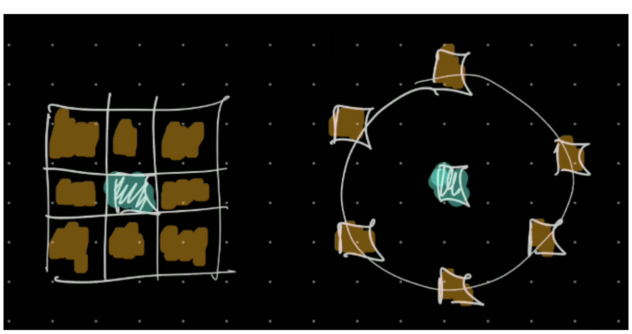
przed wykonaniem algorytmu punkty informacja o punktach jest losowo rozrzucana po macierzy - tworzac przypadkowe short-cuty

4.1.1 Local Noise



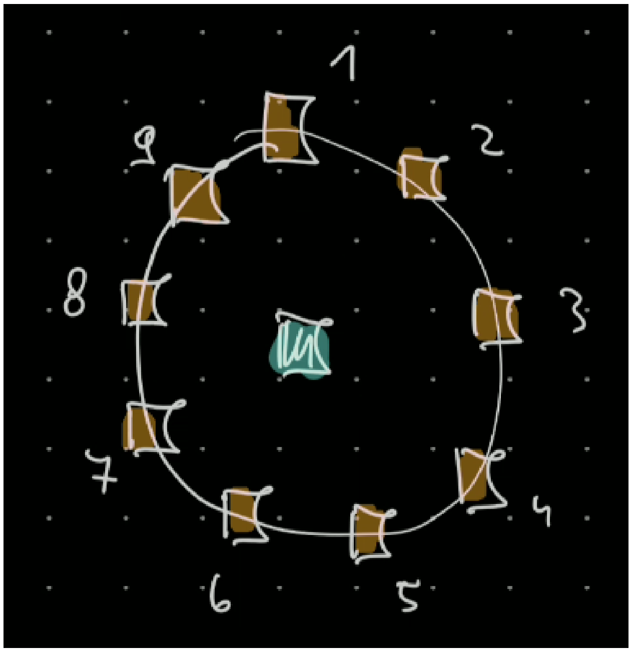
czy jest mozliwe losowanie szumu w takich plamach? aby lokalne seedy losowaly lokalne seedy? - albo probuja modyfikowac wylosowany - hehe - czyli prostujac jestem na pozycji (x,y) wylosowalem punkt (a,b) rzucam nim w okolice (a+rand(density), b+rand(density)

4.2 Anchor Type



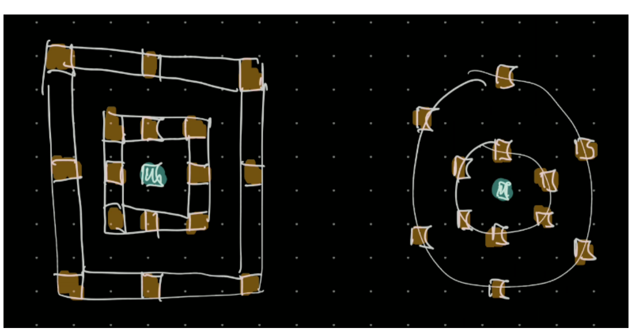
orginalnalnie jest to kwadrat 3x3, my proponujemy okrag (rownomierny), lub losowane punkty na okregu (doimplementowac).

4.2.1 Anchor Num



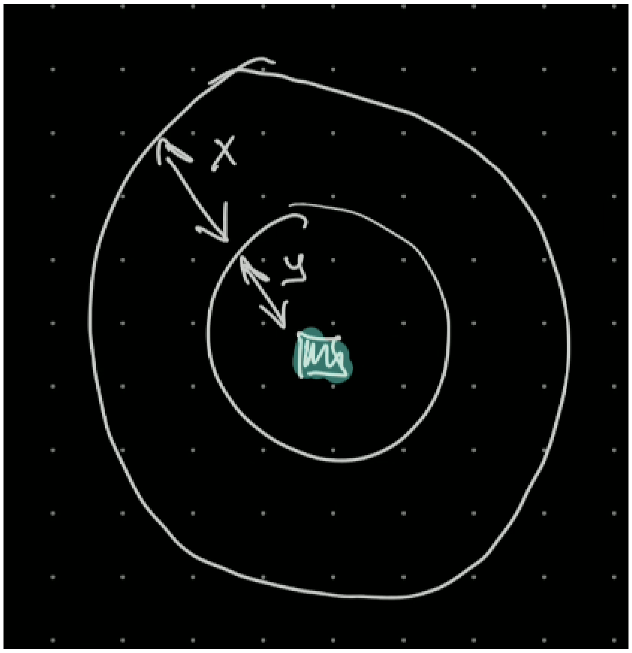
tylko dla okregow? ile ma byc punktow w anchorze (np. na okregu). czy da sie to sensownie zaimplementowac dla kwadratu? tak naprawde tylko 3x3, 5x5 maja jakikolwiek sens

4.3 Anchor Double



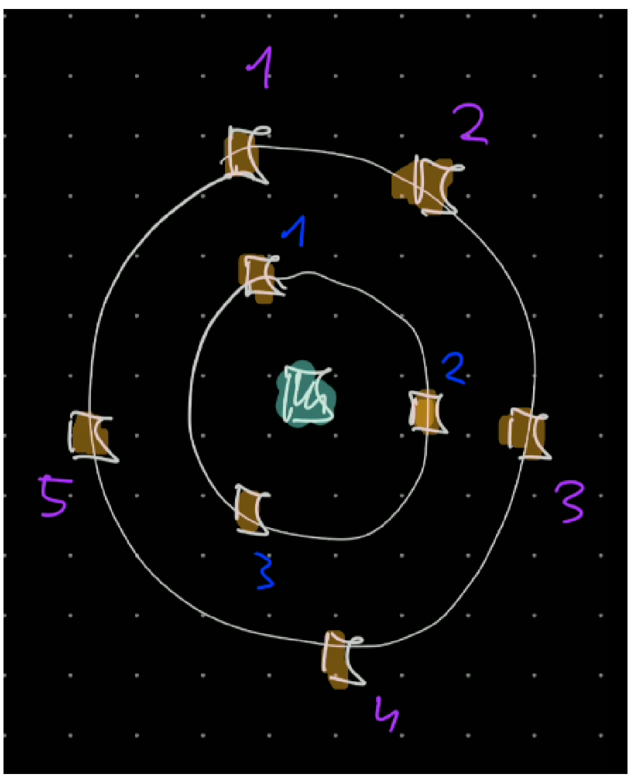
oprocz pojedynczego anchora, mozliwe jest uzycie podwojnej warstwy anchorow (czyli np. male kolko i wieksze) - idea - male kolko jest dokladne - a wielkie jest skautujace (w sumie to podobny mechanizm jak w Lookahead - wolny/szybki)

4.3.1 Anchor Distance Ratio



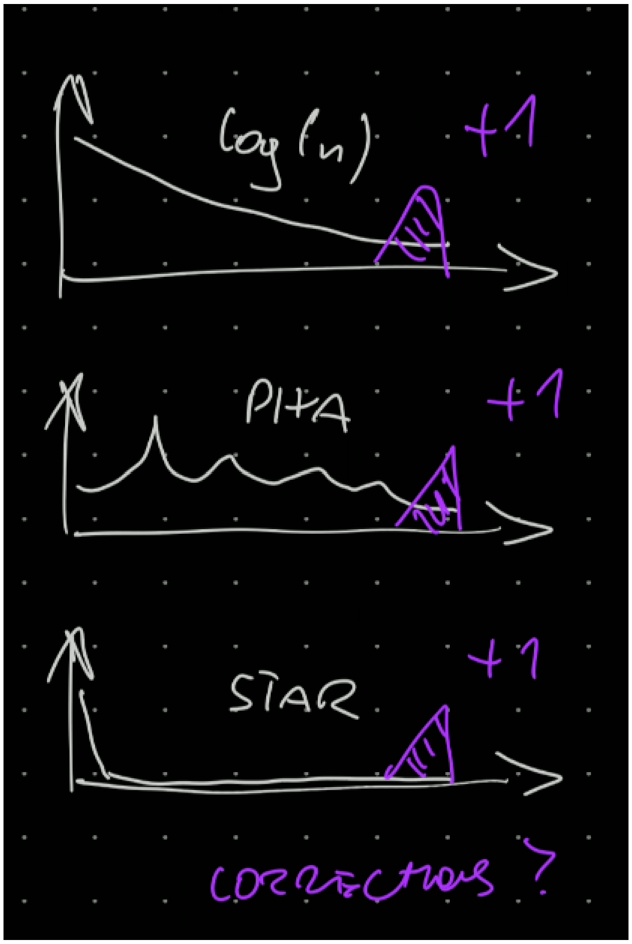
ratio pomiedzy zewnetrznymi a wewnetrznymi (doimplementowac) - mozliwosc od 0 do 1

4.3.2 Anchor Number Ratio



ratio ale ilosciowe pomiedzy zewnetrznymi a wewnetrznymi (doimplementowac) - czyli np. 5 na zewnetrznym a na bliskim 10

4.4 Step Function



aktualnie 3 warianty, standart JFA, zmieniona podstawa na 3, oraz logstar krokow, powinna istniec tez wersja 4-ta, uzalezniona od num oraz shape - pozwalajaca na robienie pilokształtnych lub isogaussowaskich ciagow.

wielomian z 3 parametrami (uzalezniony od dwoch wejsci - shape, num)

4.4.1 Correction???

mozna domimplementowac flage dla +1, +2 (jak u JFA), lub przechowywanie 2 najlepszych i ich przekazywanie (pewnie duzo wolniejsze wiec sie nie oplaca)

5 Results

przeniesc legende? JAKO OSOBNY PDF? i podac w tej sekcji - tak sie nie robi ale bylo by ok i czytelnie + wiecej miejsca na wykresy a przypadkow bedzie wiecej

5.1 Performance

wykres zostal zrobiony poprzez posortowanie scorow - dzieki temu widac roznice w przyroscie i latwo dostrzec ktory algorytm ma najwyzszy score lub jaka ma chaktersytyke (np. jest bardzo skuteczny dla waskiej grupy przykladow)

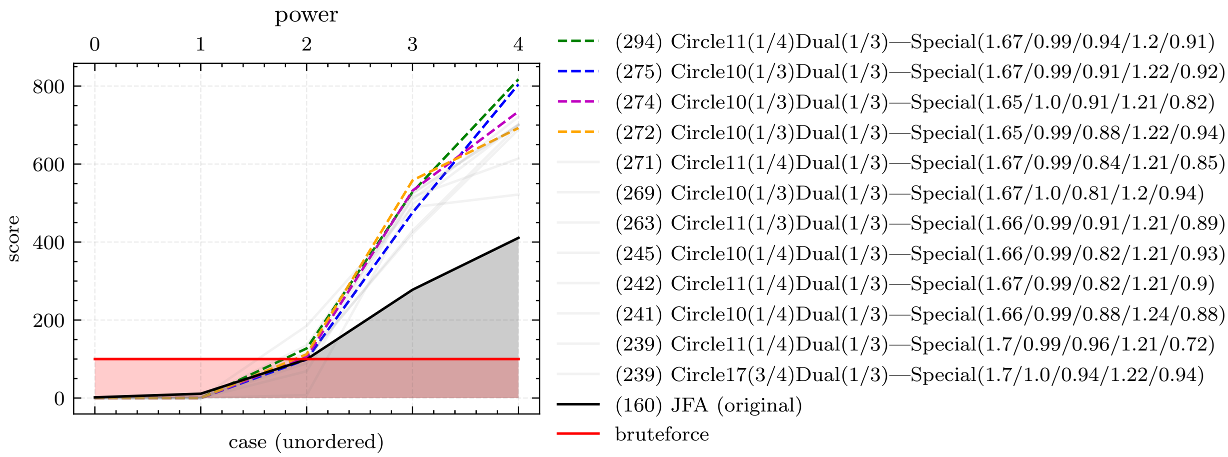


Figure 1: bla bla bla

5.2 Performance? ale taki inny

te same dane jak w poprzedniej sekcji tylko nie posortowane - dlatego widac jak wygladaja "gorki" dla danych shapow i rosnacych gestosci

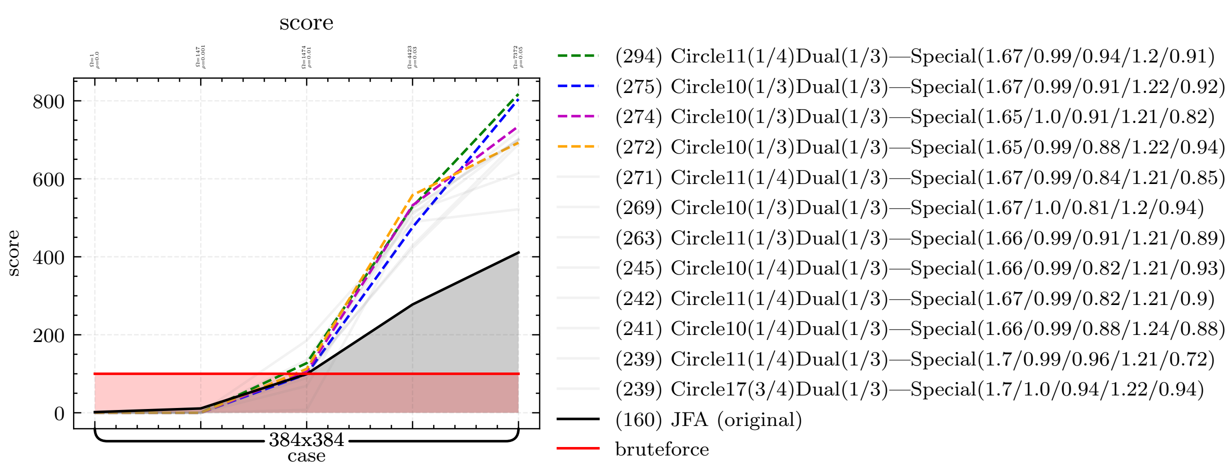


Figure 2: bla bla bla

Algorithm	ρ=0.0	ρ=0.001	ρ=0.01	ρ=0.03	ρ=0.05	Avg. score
Circle11(1/4)Dual(1/3)|Special(1.67/0.99/0.94/1.2/0.91)	0	0	**127.3**	**528.6**	**817.4**	294
Circle10(1/3)Dual(1/3)|Special(1.67/0.99/0.91/1.22/0.92)	0	0	99.0	**475.6**	**804.5**	275
Circle10(1/3)Dual(1/3)|Special(1.65/1.0/0.91/1.21/0.82)	0	0	**105.2**	**531.9**	**734.8**	274
Circle10(1/3)Dual(1/3)|Special(1.65/0.99/0.88/1.22/0.94)	0	0	**112.3**	**558.3**	**692.5**	272
Circle11(1/4)Dual(1/3)|Special(1.67/0.99/0.84/1.21/0.85)	0	0	**135.2**	**524.6**	**699.7**	271
Circle10(1/3)Dual(1/3)|Special(1.67/1.0/0.81/1.2/0.94)	0	0	**116.4**	**509.0**	**721.4**	269
Circle11(1/3)Dual(1/3)|Special(1.66/0.99/0.91/1.21/0.89)	0	0	89.4	**526.1**	**700.9**	263
Circle10(1/4)Dual(1/3)|Special(1.66/0.99/0.82/1.21/0.93)	0	0	98.3	**429.1**	**699.4**	245
Circle11(1/4)Dual(1/3)|Special(1.67/0.99/0.82/1.21/0.9)	0	0	**104.5**	**421.5**	**688.6**	242
Circle10(1/4)Dual(1/3)|Special(1.66/0.99/0.88/1.24/0.88)	0	0	7.5	**496.8**	**702.2**	241
Circle11(1/4)Dual(1/3)|Special(1.7/0.99/0.96/1.21/0.72)	0	0	68.9	**516.2**	**614.2**	239
Circle17(3/4)Dual(1/3)|Special(1.7/1.0/0.94/1.22/0.94)	0	0	**185.2**	**489.3**	**521.9**	239
Circle10(1/3)Dual(1/3)|Special(1.67/1.0/1.0/1.21/0.94)	0	0	64.5	**498.4**	**615.8**	235
Circle10(3/4)Dual(1/3)|Special(1.66/0.99/0.98/1.24/0.89)	0	0	**155.1**	**455.6**	**534.2**	228
Circle16(2/3)Dual(3/4)|Special(1.73/1.0/0.92/1.22/0.92)	0	0	**184.9**	**464.4**	**494.5**	228
Circle11(1/3)Dual(3/4)|Special(1.67/0.99/0.93/1.2/0.74)	0	0	**197.4**	**441.0**	**498.2**	227
Circle11(1/3)Dual(1/4)|Special(1.68/0.99/0.99/1.21/0.9)	0	0	0.0	**477.1**	**639.7**	223
Circle18(1/4)Dual(1/3)|Special(1.66/1.0/0.92/1.21/0.93)	0	0	**159.8**	**390.9**	**566.0**	223
Circle8(2/3)|Special(1.73/1.0/0.97/1.21/0.76)	0	0	**203.1**	**447.3**	**451.9**	220
Circle6(1/4)Dual(1/3)|Special(1.67/1.0/0.86/1.21/0.84)	0	0	0.0	**615.0**	**480.1**	219
Square|Special(1.65/0.99/0.87/1.24/0.89)	0	0	**172.5**	**382.1**	**536.8**	218
Circle10(1/3)Dual(1/3)|Special(1.69/0.99/0.98/1.22/0.95)	0	0	0.0	**478.0**	**611.1**	217
Circle8(1/4)|Special(1.66/0.99/0.97/1.22/0.88)	0	0	**203.6**	**380.7**	**490.6**	214
Circle18(3/4)Dual(1/3)|Special(1.71/1.0/0.98/1.21/0.9)	0	0	**183.2**	**369.6**	**510.1**	212
Circle11(1/4)Dual(1/3)|Special(1.66/1.0/0.81/1.24/0.89)	0	0	16.5	**493.2**	**550.0**	211
Circle10(1/3)Dual(2/3)|Special(1.71/1.0/0.93/1.21/0.88)	0	0	**226.5**	**453.5**	**366.2**	209
Circle10(1/3)|Special(1.67/0.99/0.84/1.24/0.89)	0	0	**179.1**	**383.2**	**477.8**	208
Circle10(1/3)Dual(1/3)|Special(1.66/0.99/0.93/1.24/0.69)	0	0	25.4	**457.8**	**554.1**	207
Circle15(1/3)Dual(1/3)|Special(1.67/0.99/0.96/1.21/0.67)	0	0	0.0	**405.5**	**631.6**	207
Circle10(1/4)Dual(2/3)|Special(1.72/0.99/0.85/1.21/0.85)	0	0	**192.3**	**463.7**	**378.4**	206
Circle11(1/4)Dual(1/3)|Special(1.68/0.99/0.99/1.24/0.68)	0	0	0.0	**496.1**	**537.7**	206
Circle18(1/4)Dual(2/3)|Special(1.69/1.0/0.93/1.22/0.79)	0	0	**174.5**	**361.4**	**494.0**	205
Circle10(1/3)Dual(3/4)|Special(1.67/0.99/0.9/1.2/0.88)	0	0	**200.4**	**400.9**	**425.7**	205
Circle9(1/3)|Special(1.67/0.99/0.89/1.2/0.71)	0	0	**170.1**	**344.1**	**506.2**	204
Circle10(1/3)Dual(1/3)|Special(1.72/1.0/0.96/1.22/0.9)	0	0	0.0	**478.3**	**540.6**	203
Circle9(2/3)|Special(1.65/0.99/0.94/1.23/0.95)	0	0	**172.6**	**362.1**	**480.0**	202
Circle10|Special(1.71/1.0/0.86/1.22/0.92)	0	0	**170.4**	**374.8**	**467.9**	202
Circle13(3/4)Dual(1/4)|Special(1.7/1.0/0.88/1.25/0.85)	0	0	**205.0**	**532.1**	**271.6**	201
Circle15(1/3)Dual(1/3)|Special(1.67/1.0/0.84/1.23/0.76)	0	0	0.0	**398.0**	**598.7**	199
Circle9(1/4)|Special(1.65/0.99/0.94/1.22/0.83)	0	0	**178.0**	**344.4**	**441.5**	192
Square|Special(1.71/0.99/0.95/1.21/0.74)+Noise	15	25.4	**147.8**	**333.6**	**428.6**	190
Square|Factor3+Noise	1.9	13.2	**120.4**	**301.0**	**510.5**	189
Square|Special(1.69/1.0/0.94/1.21/0.7)+Noise	15.5	28.1	**140.8**	**323.7**	**437.1**	189
Circle14(3/4)Dual(1/4)|Special(1.65/0.99/0.92/1.23/0.69)+Noise	15	0	**157.8**	**341.9**	**428.4**	188
Circle8(1/4)|Special(1.72/0.99/0.89/1.2/0.94)	0	0	**184.7**	**384.3**	**362.6**	186
Circle14(1/3)Dual|Special(1.71/1.0/0.9/1.22/0.66)+Noise	15.4	0	**122.8**	**368.7**	**423.2**	186
Circle14(3/4)Dual(1/3)|Special(1.65/0.99/0.82/1.22/0.89)	0	0	**197.2**	**362.3**	**357.1**	183
Circle11(2/3)Dual(1/3)|Special(1.7/0.99/0.92/1.21/0.94)	0	0	**235.7**	**436.8**	**238.4**	182
Circle11(2/3)Dual(1/3)|Special(1.7/1.0/0.95/1.22/0.79)	0	0	**166.0**	**408.1**	**333.8**	181
Circle11(3/4)Dual(1/4)|Default	0	15.1	**110.4**	**302.0**	**455.6**	176
Circle18(2/3)Dual(1/3)|Special(1.72/1.0/0.88/1.21/0.69)+Noise	14.5	0	**124.9**	**366.6**	**370.2**	175
Circle17(3/4)Dual|Special(1.66/1.0/0.95/1.22/0.9)	0	0	**201.9**	**415.5**	**256.1**	174
Circle10(2/3)|Special(1.69/0.99/0.91/1.2/0.71)	0	0	**164.5**	**366.5**	**340.5**	174
Circle12(1/3)|Special(1.7/1.0/0.8/1.21/0.94)	0	0	**161.8**	**299.2**	**386.3**	169
Circle6(1/4)Dual(1/3)|Special(1.68/1.0/0.92/1.2/0.94)+Noise	18.6	0	29.3	**513.1**	**275.9**	167
Circle11(3/4)|Special(1.67/1.0/0.83/1.21/0.73)	0	0	**150.0**	**309.8**	**340.6**	160
Circle11(1/3)Dual(3/4)|Special(1.67/0.99/0.9/1.22/0.79)	0	0	**217.4**	**430.2**	**152.0**	159
Circle17(1/4)|Special(1.73/1.0/0.84/1.22/0.76)	0	0	**135.6**	**280.4**	**376.5**	158
Circle16(3/4)Dual(3/4)|Special(1.7/1.0/0.87/1.25/0.8)+Noise	17.1	0	**169.4**	**391.2**	**206.8**	156
Circle16(3/4)Dual(3/4)|Default	1.7	11.8	93.6	**256.8**	**395.4**	151
Circle13(3/4)Dual(1/3)|Special(1.69/1.0/0.96/1.2/0.89)+Noise	17.3	0	**198.8**	**383.8**	**138.4**	147
Square|Default+Noise	1.3	9	82.0	**228.8**	**361.8**	136
Circle16(3/4)Dual(2/3)|Special(1.7/0.99/0.92/1.24/0.92)	0	0	**207.7**	**471.1**	0.0	135
Circle11(3/4)Dual(1/3)|Special(1.67/1.0/0.85/1.21/0.94)	0	0	**237.5**	**434.4**	0.0	134
Circle10(1/4)Dual(2/3)|Special(1.67/0.99/0.8/1.21/0.9)+Noise	18.1	0	**193.7**	**249.0**	0.0	92

5.3 Error Rate

score jest scisle powiazany z errorem ale tutaj przejrzyscie widac w jakich przypadkach jest naprawde zle i ile (szarych) czesto jest bardzo szybkich - ale rowniez bardzo blednych dlatego maja finalnie niskie scory

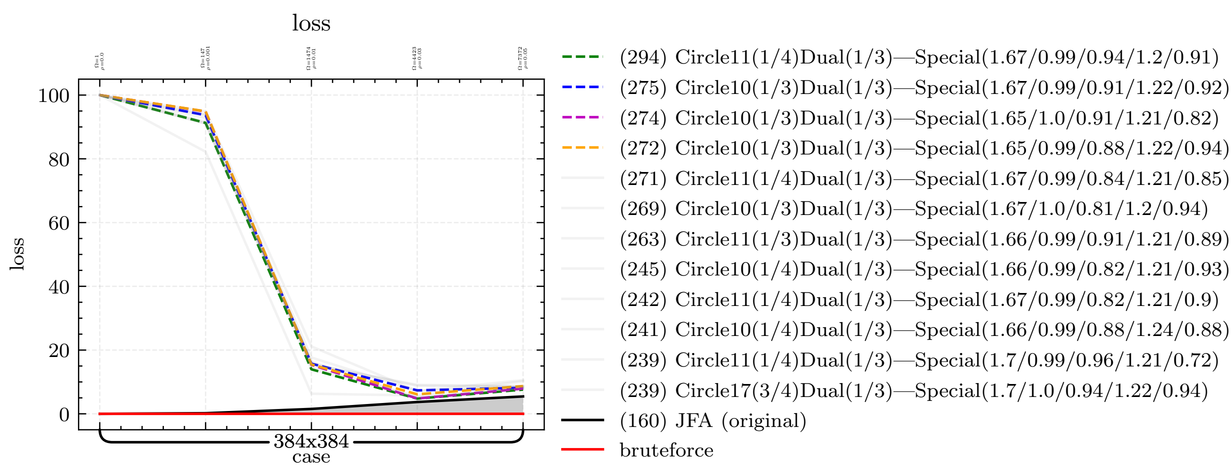


Figure 3: bla bla bla

5.4 Objectives

naprawic generowanie tego wykresu

czyli co ma wplyw na co (w sumie to najwazniejsze mialo byc w pracy)

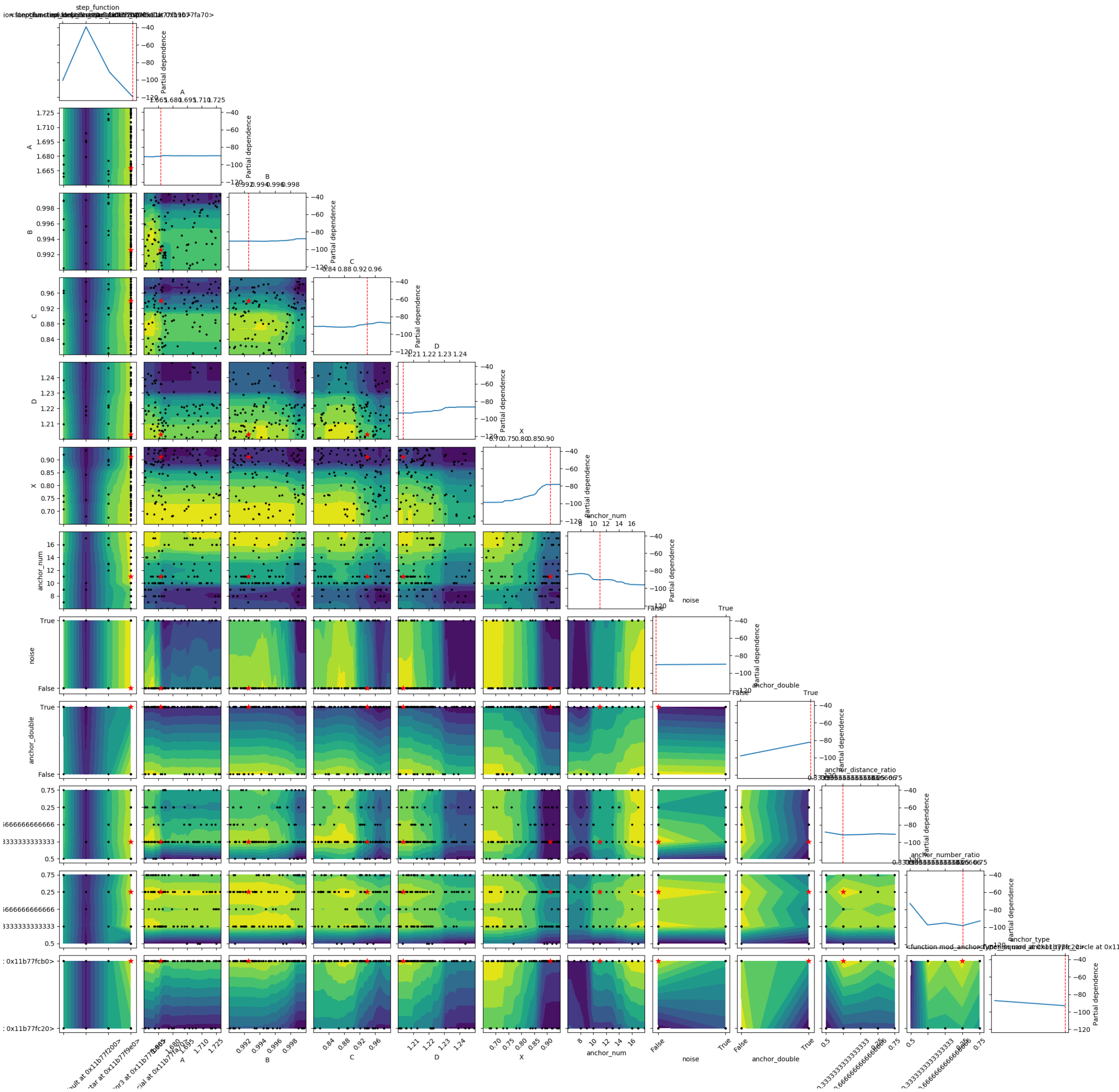


Figure 4: bla bla bla

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 [7], it remains challenging to balance the workload in such an approach. *Lord 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 [3].

8 Acknowledgements

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

References

[1] Kenneth E Hoff III et al. "Fast computation of generalized Voronoi diagrams using graphics hardware". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, pp. 277–286.

[2] Ian Fischer and Craig Gotsman. "Fast approximation of high-order Voronoi diagrams and distance transforms on the GPU". In: *Journal of Graphics Tools* 11.4 (2006), pp. 39–60.

[3] Guodong Rong and Tiow-Seng Tan. "Jump flooding in GPU with applications to Voronoi diagram and distance transform". In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 2006, pp. 109–116.

[4] Avneesh Sud et al. "Interactive 3D distance field computation using linear factorization". In: *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. 2006, pp. 117–124.

[5] Guodong Rong and Tiow-Seng Tan. "Variants of jump flooding algorithm for computing discrete Voronoi diagrams". In: *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)*. IEEE. 2007, pp. 176–181.

[6] Jens Schneider, Martin Kraus, and Rüdiger Westermann. "GPU-based real-time discrete Euclidean distance transforms with precise error bounds." In: *VISAPP (1)*. 2009, pp. 435–442.

[7] Thanh-Tung Cao et al. "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*. 2010, pp. 83–90.

[8] Francisco de Assis Zampirolli and Leonardo Filipe. "A fast CUDA-based implementation for the Euclidean distance transform". In: *2017 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE. 2017, pp. 815–818.

[9] Takumi Honda et al. "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. 2017, pp. 362–371.

[10] Manduhu Manduhu and Mark W Jones. "A work efficient parallel Algorithm for exact Euclidean distance transform". In: *IEEE Transactions on Image Processing* 28.11 (2019), pp. 5322–5335.