



Politechnika Wrocławskiego

**Wydział Matematyki**

Kierunek studiów: Matematyka stosowana

Specjalność: –

Praca dyplomowa – inżynierska

**GENEROWANIE OBRAZÓW  
W OPARCIU O PARKIETAŻE**

Filip Oszczepaliński

słowa kluczowe:  
próbkowanie dysków Poissona, algorytm Bridsona, metoda relaksacji, minimalny błąd cięcia granicznego, jądro filtra obrazu, kafelki Wanga, jądro gaussowskie

krótkie streszczenie:

W pracy zaproponowano wydajną metodę generowania niejednorodnego szumu na płaszczyźnie. Za pomocą algorytmu Bridsona oraz metody relaksacji wygenerowano jednorodny szum niewielkiego rozmiaru stanowiący podstawę do utworzenia kafelków Wanga, z których można ułożyć parkietaż dowolnie dużej powierzchni. Niejednorodność szumu zagwarantowano stosując metodę akceptacji i odrzuceń na ułożonym parkietażu. Przedstawiono przykład zastosowania zaproponowanego algorytmu do fotografii cyfrowej.

Opiekun pracy dyplomowej	dr inż. Andrzej Giniewicz	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	ocena	podpis

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:\**

a) kategorii A (akta wieczyste)

b) kategorii BE 50 (po 50 latach podlegające ekspertryzje)

\* niepotrzebne skreślić

pieczętka wydziałowa

Wrocław, rok 2024





Faculty of Pure and Applied Mathematics

Field of study: Applied Mathematics

Specialty: –

Engineering Thesis

## GENERATING IMAGES BASED ON PARQUETS

Filip Oszczepaliński

keywords:

Poisson disk sampling, Bridson algorithm, relaxation method, minimum error boundary cut, image filter kernel, Wang tiles, gaussian kernel

short summary:

This thesis proposes an efficient method for generating inhomogeneous noise on a plane. Using Bridson's algorithm and the relaxation method, homogeneous noise of small size was generated as the basis for creating Wang tiles from which an infinite parquet of large area can be arranged. Heterogeneity of the noise was guaranteed by applying the acceptance and rejection method on the laid parquet. An example of the application of the proposed algorithm to digital photography is presented.

Supervisor	dr inż. Andrzej Giniewicz	.....	.....
	Title/degree/name and surname	grade	signature

For the purposes of archival thesis qualified to:\*

- a) category A (perpetual files)
- b) category BE 50 (subject to expertise after 50 years)

\* delete as appropriate

stamp of the faculty

Wrocław, 2024



# Spis treści

<b>Wstęp</b>	<b>3</b>
<b>1 Teoria</b>	<b>5</b>
1.1 Próbkowanie dysków Poissona . . . . .	6
1.2 Relaksacja . . . . .	6
1.3 Kafelki Wanga . . . . .	7
1.4 Minimalny błąd cięcia granicznego . . . . .	8
1.5 Jądro filtra obrazu . . . . .	11
1.6 Konwertowanie zapisu kolorów . . . . .	12
1.6.1 Konwersja z RGB do HSL . . . . .	12
1.6.2 Konwersja z HSL do RGB . . . . .	13
1.7 Jądro gaussowskie . . . . .	14
1.8 Dystrybuanty i funkcje . . . . .	14
1.9 Działania na macierzach . . . . .	14
<b>2 Implementacja</b>	<b>15</b>
2.1 Tworzenie jednorodnych próbek szumu . . . . .	15
2.2 Tworzenie kafelków szumu . . . . .	16
2.2.1 Pobieranie segmentów szumu z przestrzeni . . . . .	17
2.2.2 Tworzenie pełnego zestawu kolorów kafelków . . . . .	18
2.2.3 Tworzenie kafelka . . . . .	18
2.3 Generowanie jednorodnego szumu obrazu . . . . .	23
2.4 Tworzenie niejednorodnego szumu . . . . .	25
2.5 Nakładanie szumu na obraz . . . . .	25
<b>3 Wyniki własne</b>	<b>29</b>
<b>Podsumowanie</b>	<b>37</b>
<b>Bibliografia</b>	<b>38</b>



# Wstęp

Początki fotografii datowane są na XIX wiek. W fotografii analogowej podczas robienia zdjęcia obraz zostaje najczęściej zapisany na błonie fotograficznej, która zawiera emulsję światłoczułą. Emulsja światłoczuła to sole srebra zawieszone w żelatynie. Gdy zostaje otwarta przysłona, światło wpada przez obiektyw wprost na błonę. Foton uderza w cząsteczki soli srebra, krystalizując je. W ten sposób otrzymuje się obraz ukryty. Aby uzyskać widoczny obraz, trzeba w ciemni zanurzyć obraz w wywoływacz chemicznym. Podczas zachodzącej reakcji kryształy soli srebra zamieniają się w czarne metaliczne srebro. Po zakończeniu procesu wywoływania otrzymuje się negatyw obrazu przedstawiony na rysunku 1, w którym miejsca na które padło najwięcej fotonów, mają najwięcej cząsteczek srebra. To, co na obrazie powinno być jasne, na negatywie jest ciemne i na odwrót. Szum na zdjęciu powodują zabłakane fotony, w momencie padania na emulsję światłoczułą. Zazwyczaj światło obrysowuje fotografowany obiekt, jednak może się zdarzyć, że foton odbije się od jakiegoś punktu i zmieni swoją trajektorię, co spowoduje szum. W pracy zaprezentowano algorytm, efektem którego był szum wyglądającym przypominającym szum uzyskany w fotografii analogowej.

W artykule, który był inspiracją do niniejszej pracy, nakładany był niebieski szum na obraz [9]. Szum ten był generowany za pomocą modelu logicznego na cały obraz, przy użyciu dwóch sposobów. Autorzy wykonali obliczenia, by wyznaczyć, który sposób będzie działał szybciej względem parametrów szumu. Ich algorytm musiał wygenerować na całą przestrzeń model logiczny, po czym wykorzystując metodę Monte Carlo, poruszał przestrzeń wypełnioną dyskami po obrazie i zliczał, ile razy dany piksel znajdował się w zasięgu dysku. Na koniec wszystko dzielono przez liczbę wykonanych ruchów i uzyskiwano wynik. Dla dużych zdjęć algorytm generowania szumu działa zbyt wolno<sup>1</sup>.

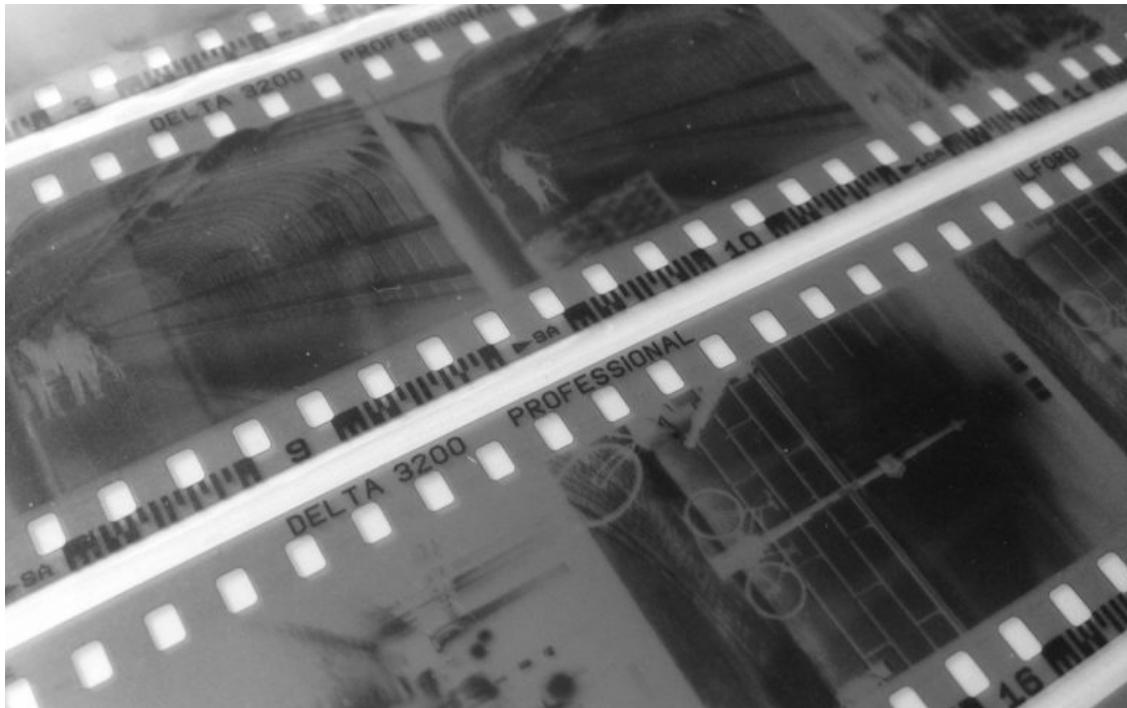
W niniejszej pracy został zaprezentowany sposób zastosowania parkietu do generowania szumu na cały obraz z małych próbek szumu. Próbki zostały pobrane z jednorodnego szumu wygenerowanego za pomocą próbkowania dysków Poissona na małej płaszczyźnie, a następnie połączone używając metody minimalnego błędu cięcia granicznego oraz jądra filtra obrazu. Proces generowania szumu z wykorzystaniem parkietu wykonywał się szybciej niż gdyby generowano szum od razu na obraz. Do wykonania parkietu zostały wykorzystane kafelki Wanga. Metoda ta polega na układaniu płaszczyzny kwadratami z kolorowymi ścianami, tak aby ściany przylegające były do siebie dopasowane. Za pomocą tej metody można generować nieskończoną wielkością płaszczyznę. Po wygenerowaniu szumu na cały obraz za pomocą jasności piksela została podjęta decyzja czy dany szum, który ma być nałożony na piksel, zostanie zaakceptowany lub odrzucony.

W rozdziale 1 przedstawiono działania, których celem jest niejednorodny szum nałożony na obraz względem jasności, oraz opis zagadnień teoretycznych, z zakresu tworzenia szumów, tworzenie kafelków szumu, generowanie szumu na płaszczyznę za pomocą parkie-

---

<sup>1</sup><https://github.com/darktable-org/darktable/issues/4451#issue-578656162>

tażu oraz tworzenie niejednorodnego szumu. Rozdział 2 zawiera propozycję implementacji autorskiego algorytmu wykorzystującego parkietaż do generacji szumu wraz z diagramami wizualizującymi implementacje. Przykładowe zastosowanie algorytmu zostało zaprezentowane na wybranych obrazach w rozdziale 3. Cały program został napisany w języku Python z wykorzystaniem bibliotek: `Pillow`, `Numpy`, `Scipy`, `Scipy.Stats` i `sys_color`.



Rysunek 1: Negatyw obrazu [10].

# Rozdział 1

## Teoria

Generowanie szumu dla całego obszaru  $N \times M$  za pomocą próbkowania dysków Poissona trwało zbyt długo. W celu przyspieszenia procesu zastosowano parkietaż. Wykorzystując jasność piksela, uzyskano niejednorodny szum. Poniżej znajduje się opis kroków zaproponowanego algorytmu wraz z odnośnikami do miejsc w pracy, w których znajduje się ich dokładny opis.

1. Generowanie jednorodnego szumu na siatkę  $n \times n$ , gdzie  $0 < n << \min\{N, M\}$ , za pomocą próbkowania dyskami Poissona. Szumy zapisano w macierzy binarnej, gdzie 1 to szum, a 0 to brak szumu. Szczegółowy opis został przedstawiony w podrozdziale 1.1.
2. Pobranie losowo czterech nienachodzących na siebie segmentów z szumu  $n \times n$  o rozmiarach  $m \times m$ , gdzie  $0 < m < \frac{n}{2}$  i  $m$  jest liczbą całkowitą parzystą.
3. Do czterech wybranych segmentów  $m \times m$  zostały dopasowane kolory, po czym zdefiniowano zestaw kolorów kafelków oraz ile kafelków miało zostać wybranych z zestawu. Następnie stworzono kafelki, łącząc ze sobą segmenty przypisane do kafelka, pobrano obszary nakładania się segmentów i nakładano na nie zdefiniowane jądro. Na podstawie zmodyfikowanych segmentów wyznaczano ścieżkę ich łączenia, wykorzystując metodę minimalnego błędu cięcia granicznego. Po połączeniu 4 segmentów otrzymano romb, z którego wycięto kwadrat, który stawał się kafelkiem. Wizualizacja rombu oraz kafelka została pokazana na rysunku 1.3. Szczegółowy opis znajduje się w podrozdziale 1.4 oraz 1.5.
4. Z kafelków został utworzony parkietaż o rozmiarach równych lub większych niż  $N \times M$ , wykorzystując stochastyczne kafelki Wanga. Jeśli parkietaż był większy, to został docięty do rozmiarów  $N \times M$ . Szczegółowy opis znajduje się w podrozdziale 1.3.
5. Utworzenie rozmazanego obrazu za pomocą jądro gaussowskie celem uzyskania informacji o otoczeniu piksela. Następnie pobrano piksel, który pokrywa się z jednorodnym szumem, w zapisie RGB, po czym sformatowano go do zapisu HSL, który umożliwił wyciągnięcie wartości jasności. Użyto metody akceptacji-odrzucenia, wykorzystującej dystrybuanty, by zmniejszać ilość pojawiającego się szumu wraz ze wzrostem jasności obszaru. Po nałożeniu szumu na piksel, piksel z zapisu HSL przechodził do zapisu RGB. Szczegółowy opis został przedstawiony w podrozdziale 1.6–1.8.

## 1.1 Próbkowanie dysków Poissona

Za pomocą próbkowania dysków Poissona został wygenerowany jednorodny szum, czyli szum na całej płaszczyźnie bez uwzględniania jasności piksela.

**Definicja 1.1.** W zbiorze próbek dysków Poissona, dystans pomiędzy dwiema próbками nie może być mniejszy niż określona odległość. Odległość jest zdefiniowana przez promień  $r$  dysku Poissona, który jest minimalną odległością między dwoma centrami próbek.

**Algorytm 1.2** (Algorytm Bridson). Algorytm Bridsona został wynaleziony przez Roberta Bridsona i służy do generowania jednorodnych próbek dysków Poissona [2]. Sposób działania algorytmu wygląda następująco:

1. Utwórzmy  $n$ -wymiarową siatkę tła dla przechowywania próbek i przyspieszania przestrzennego wyszukiwania. Rozmiar komórki jest równy  $\frac{r}{\sqrt{n}}$ , tak aby każda komórka zawierała maksymalnie jedną próbke.
2. Wybieramy początkową próbkę  $x_0$  losowo wybraną równomiernie z dziedziny. Następnie umieszcimy próbkę w siatce tła i stworzymy listę aktywnych punktów, gdzie wartość pierwszego elementu jest równa współrzędnym centra  $x_0$ .
3. Dopóki lista aktywnych punktów nie jest pusta, wybieramy losową próbkę  $(x_i)$  z listy. Wygenerujemy  $k$  punktów wybranych w sposób jednolity ze sferycznego pierścienia pomiędzy promieniem  $r$  a  $2r$  wokół  $x_i$ . Dla każdej nowo wygenerowanej próbki sprawdzamy, czy w obrębie jej promienia znajduje się inna próbka, używając w tym celu siatki tła. Jeśli nowo wygenerowana próbka nie koliduje z żadnym ze swoich sąsiadów, to akceptujemy ją oraz dodajemy ją do listy aktywnych punktów. Jeśli po  $k$  podejściach nie znajdziemy żadnej nowej próbki dla  $x_i$  to usuń  $x_i$  z listy aktywnych punktów.

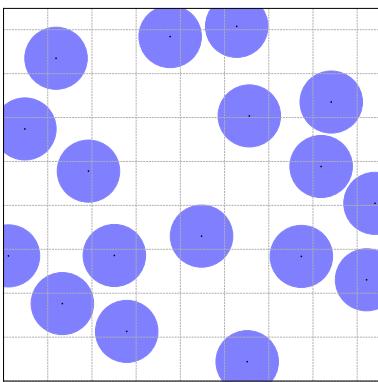
Końcowy możliwy wynik jest ukazany na rysunku 1.1.

## 1.2 Relaksacja

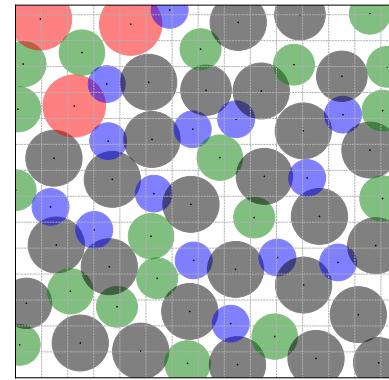
Metoda relaksacji gwarantuje, że zostanie uzyskana odpowiednia ilość dysków na danej płaszczyźnie za pomocą próbkowania dysków Poissona.

**Definicja 1.3** (Metoda relaksacji). Gdy na płaszczyźnie zabraknie miejsca na nowe dyski o promieniu  $r$ ,  $r$  jest mnożony przez stałą  $\alpha \in (0,1)$  [8].

Najczęściej przyjmuje się wartości blisko 1, takie jak 0,99, 0,97. W niniejszej pracy metoda relaksacji została połączona z algorytmem Bridsona przedstawionym w algorytmie 1.2. Oprócz zmniejszania promienia zwiększano także ilość  $k$  prób znalezienia nowej próbki. Wizualizacja końcowego wyniku pokazano na rysunku 1.2. Szum, który został stworzony za pomocą tych metod, zapisano w postaci macierzy binarnej, gdzie 1 to szum, a 0 to brak szumu.



Rysunek 1.1: Próbkowanie dysków Poissona dla algorytmu Bridsona.



Rysunek 1.2: Wizualizacja metody relaksacji dla próbkowania dysków Poissona.

### 1.3 Kafelki Wanga

Kafelki Wanga pozwalają generować płaszczyznę wypełnioną jednorodnym szumem, wykorzystując kafelki wygenerowane w poprzez działania wykonane w rozdziale 1.4. Przy układaniu ścian, należy zadeklarować, jakie kolory mają mieć ściany w poziomie i pionie. W pracy, do kolorów zostały dopasowane 4 segmenty pobrane z macierzy szumu wygenerowanej w rozdziale 1.1. Pojęcie te zaproponował Hao Wang w 1961 roku. Następnie przechodzi się do układania ich na płaszczyźnie.

**Definicja 1.4** (Kafelki Wanga). Kafelki Wanga to kafelki o kształcie wielokątu z kolorowymi krawędziami. Mogą one być ze sobą sąsiadami tylko wtedy, gdy ich przylegające ściany mają ten sam kolor. Kafelków nie można rotować, ani lustrzanie odbijać [3].

W pracy zastosowano stochastyczne kafelki Wanga, ponieważ podjęto próbę ograniczenia okresowości podczas kładzenia kafelków.

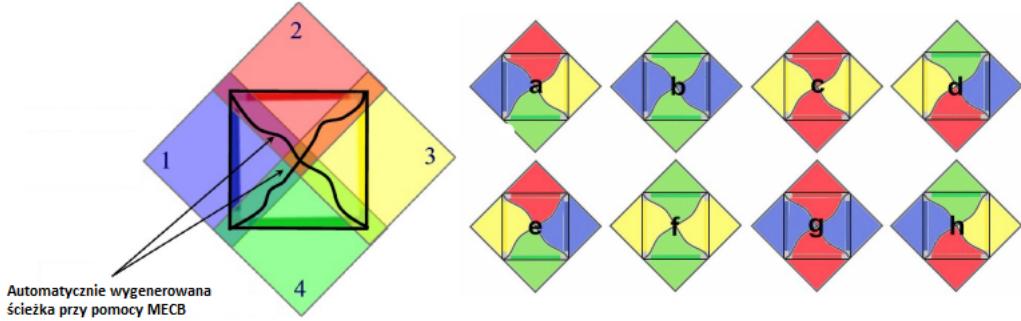
**Definicja 1.5** (Stochastyczne kafelki Wanga). Podczas układania kafelków na płaszczyźnie wybiera się losowo kafelek z wyznaczonego zbioru kafelków, który może być sąsiadem względem poprzedniego kafelka [3].

**Uwaga 1.6.** Stochastyczne kafelki Wanga mogą nie być aperiodyczne.

**Algorytm 1.7.** Dane wejściowe to zbiór kafelków Wanga.

1. Wybieramy jakikolwiek kafelek dla lewego górnego narożnika.
2. Uzupełniamy pierwszy wiersz, dobierając kafelek tak, aby kolor lewej ściany był równy kolorowi prawej ściany poprzedniego kafelka.
3. Pierwszy kafelek kolejnego rzędu dobieramy losowo tak, aby kolor górnej ściany był taki sam jak kolor dolnej ściany kafelka położonego nad nim.
4. Kontynuując ten sam wiersz, dobieramy losowo kafelek tak, aby lewa ściana była tego samego koloru jak prawa ściana lewego kafelku oraz górny kolor ściany był równy kolorowi dolnej ściany górnego kafelka.

5. Powtarzamy kroki 3 i 4 do momentu uzyskania wymaganego efektu.



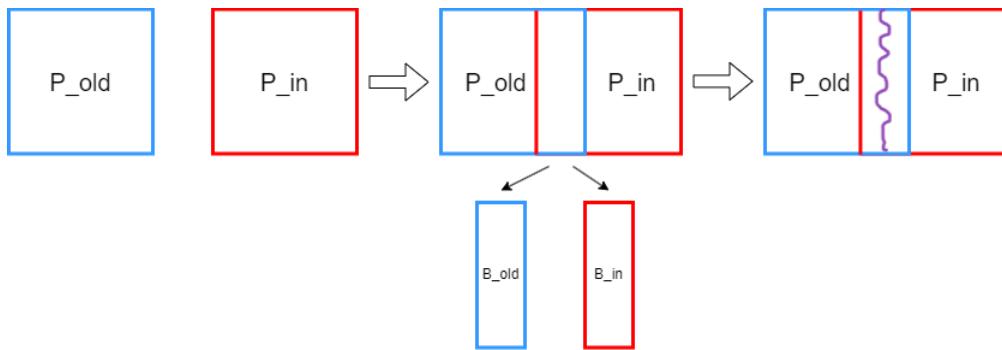
Rysunek 1.3: Kafelek Wanga [3].

Bazując na rysunku 1.3 i na niniejsze pracy, segmenty szumu są kolorowymi rombami, które zostały połączone za pomocą minimalnego błędu cięcia granicznego. Z tak uzyskanego romba wycinany jest kafelek w kształcie kwadratu. Każdy segment szumu jest dopasowany tylko do jednego koloru, by po połączeniu przylegających ścian dostać jednolitą całość. Kafelek ma rozmiar większy niż pojedynczy segment. Romb powstaje z połączenia czterech segmentów musi mieć wymiary parzyste, aby romb dało się przeciąć na cztery równe części. Wszystkie działania były robione na rombach, ponieważ podczas rotacji piksele są nadpisywane, a obraz staje się rozmazany. W rozdziale implementacji cięcie były wyznaczane na kwadratach, a następnie powstały kwadrat rotowano do pozycji rombu tak, by zachować pozycje szumu względem oryginalnego kwadratu.

## 1.4 Minimalny błąd cięcia granicznego

Za pomocą metody minimalnego błędu cięcia granicznego sklejano segmenty zawierające szum, by otrzymać kafelek. Metoda ta gwarantuje ciągłość i uniknięcie artefaktów, czyli zakłóceń obrazu. W dalszej części rozdziału segmenty były traktowane jako próbki szumu.

Niech  $P_{old}$  to poprzednio wyliczona próbka szumu.  $P_{in}$  to wybrana próbka szumu, która będzie łączona z  $P_{old}$ . Niech  $(w_p, w_o)$  to rozmiary obszarów  $B$ , które będą na siebie nachodzić podczas łączenia, w zależności od sposobu łączenia próbek  $P_{old}$  i  $P_{in}$ .



Rysunek 1.4: Wizualizacja etapów łączenia pionowego.

Poniższe działanie znajdowania najlepszej ścieżki dla minimalnego błędu cięcia granicznego zostało pokazane dla przypadku łączenia pionowego. Dla łączenia pionowego

wszystkie macierze i obszary miały rozmiar  $w_p \times w_o$ . Na początku została obliczona „różnica”, inaczej nazywana błędem, między  $B^{P_{old}}$  a  $B^{P_{in}}$ . Zamiast odejmowania zostało zastosowane dodawanie, by uniknąć sytuacji odjęcia się dwóch szumów. Wynik zapisano w postaci macierzy  $e$  by końcowo uzyskać

$$e(i, j) = B^{P_{old}}(i, j) + B^{P_{in}}(i, j).$$

Następnie wyliczono koszt budowy poszczególnych długości ścieżek. Koszty zapisano w pustej macierzy  $E$  o tych samych rozmiarach co  $e$ . Wyznaczenie kosztów ścieżek rozpoczęło się od końcowego wiersza  $E[w_p - 1]$ , który był równy końcowemu wierszowi  $e[w_p - 1]$ . Wyznaczając wartości  $E[i, j]$  dla kolejnych, czyli w tym przypadku idąc do góry, wierszy, sprawdzieni zostali ich dolni najbliżsi sąsiedzi i wybrano tego z najmniejszą wartością, czyli  $\min(E_v(i + 1, j - 1), E_v(i + 1, j), E_v(i + 1, j + 1))$ . Aby uzyskać końcowy wynik  $E[i, j]$  oprócz dodania sąsiada należało również dodać błąd  $e[i, j]$  a efekt końcowy to

$$E(i, j) = e(i, j) + \min(E(i + 1, j - 1), E(i + 1, j), E(i + 1, j + 1)). \quad (1.1)$$

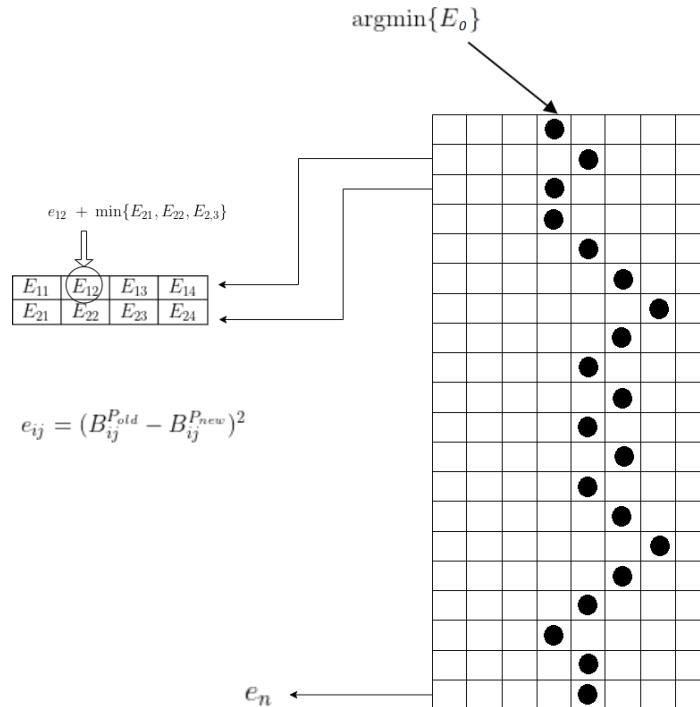
Jednocześnie, aby nie szukać ponownie dolnego sąsiada, zapisano go w pustej macierzy  $T$ , której wartości to współrzędne położenia dolnego sąsiada, którego wartość została wzięta do wyznaczenia  $E[i, j]$ , czyli

$$T(i, j) = \operatorname{argmin}(E(i + 1, j - 1), E(i + 1, j), E(i + 1, j + 1)).$$

Po wypełnieniu macierzy  $E$  i  $T$  czas przejść do znalezienia najlepszej ścieżki  $\gamma$  o długości  $w_p$ . Najpierw wyznaczono początek ścieżki, który równy był współrzędnym komórki z najmniejszą wartością w pierwszym rzędzie, ponieważ pierwszy rząd zawierał informacje o całkowitym koszcie utworzenia ścieżek o długości  $w_p$ , a następnie wykorzystując informacje o dolnych sąsiadach, zawartych w macierzy  $T$ , została odtworzona ścieżka  $\gamma$ , czyli

$$\gamma[i] = T[\gamma[i - 1]].$$

W ten sposób została otrzymana najlepsza ścieżka cięcia dla łączzenia pionowego, rysunek 1.5 pokazuje działanie i możliwy przebieg ścieżki  $\gamma$ .



Rysunek 1.5: Minimalny błąd cięcie granicy dla łączenia ścian pionowych.

Dla przypadku łączenia poziomego zmieniał się rozmiar obszarów  $B$  na  $(w_o \times w_p)$  oraz dla wszystkich pozostałych macierzy. Zamiast poruszania się w góre po wierszach, algorytm poruszał się w lewo po kolumnach. Czyli podczas obliczania  $e$  nie zmieniało się nic, natomiast w przypadku  $E$  najpierw wyznaczano ostatnią kolumnę przy pomocy ostatniej kolumny  $e$ . Potem, poruszając się w lewo, wyznaczano kolejne wartości  $E$ :

$$E(i, j) = e(i, j) + \min(E(i - 1, j + 1), E(i, j + 1), E(i + 1, j + 1)),$$

oraz

$$T(i, j) = \operatorname{argmin}(E(i - 1, j + 1), E(i, j + 1), E(i + 1, j + 1)).$$

Po wyznaczeniu  $E$  i  $T$  znajdowano początek najlepszej ścieżki  $\gamma$  spośród wszystkich ścieżek o długości  $w_p$ . Czyli została przeszukana pierwsza kolumna, aby znaleźć współrzędne dla komórki z najmniejszą wartością, po czym wykorzystując  $T$  odtworzono  $\gamma$ , w następujący sposób

$$\gamma[j] = T[\gamma[j - 1]].$$

Ostatni i najbardziej złożony przypadek to łączenie w kształcie  $L$ . Zostały wprowadzone następujące oznaczenia:  $P_{oldh}$  to poprzednio wyliczona górna próbka szumu,  $P_{oldv}$  to poprzednio wyliczona lewa próbka szumu,  $e_h, E_h, T_h$  wyliczono dla łączenia poziomego, a  $e_v, E_v, T_v$  wyliczono dla łączenia pionowego. Warto zaznaczyć, że  $P_{oldh}$  i  $P_{oldv}$  miały wspólny obszar wielkości  $w_o \times w_o$ . W tym właśnie miejscu ulokowano punkt  $(i^*, i^*)$ , który łączył najlepszą ścieżkę  $\gamma_h$  dla cięcia poziomego i  $\gamma_v$  dla cięcia pionowego. Na tym kawałku  $e_h$  i  $e_v$  były równe, zatem można było na tym obszarze wyznaczyć nowe  $E[i, i] = E_v[i, i] + E_h[i, i] - e[i, i]$  dla  $i \in \{0, 1, \dots, w_o - 1\}$ . Odjęto  $e$ , gdyż wynika to z obliczania  $E$  (1.1). Następnie zostały wyznaczone  $\gamma_h$  i  $\gamma_v$ , każda długości  $w_p - i^*$ . Punkt  $(i^*, i^*)$  został końcem ścieżki  $\gamma_v$ , czyli  $\gamma_v[i] = T_v[\gamma_v[i + 1]]$ , oraz punkt  $(i^*, i^*)$  startem ścieżki  $\gamma_h$ , czyli  $\gamma_h[j] = T_h[\gamma_h[j - 1]]$ . Pod koniec dwie ścieżki połączono i w ten sposób uzyskano najlepszą ścieżkę  $\gamma$  dla łączenia typu  $L$ .

**Algorytm 1.8** (MECB). Dane wejściowe to  $P_{in}, P_{out}, typ$ .  $typ$  oznacza rodzaj łączenia.

(I) Wyliczamy różnicę do kwadratu  $e = (B^{P_{in}} + B^{P_{old}})$ .

(II) W zależności od  $typ$ :

- Przypadek pionowy ( $w_o$  to szerokość obszaru nakładania się elementów obrazu):
  - 1) Wyliczamy skumulowany pionowy minimalny błąd  $E_v$ , gdzie  $E_v(w_p - 1, j)$ , dla  $j \in \{0, \dots, w_o - 1\}$
  - 2) Dla każdego  $i$  od  $w_p - 2$  do 0 wyliczamy:
    - $E_v(i, j) = e(i, j) + \min(E_v(i + 1, j - 1), E_v(i + 1, j), E_v(i + 1, j + 1))$ , gdzie  $j \in \{0, \dots, w_o - 1\}$ ,
    - $T_v(i, j) = \operatorname{argmin}(E_v(i + 1, j - 1), E_v(i + 1, j), E_v(i + 1, j + 1))$ .
  - 3) Definiujemy  $j^* = \operatorname{argmin}_j E_v(0, j)$ .
  - 4) Odtwarzamy ścieżkę  $\gamma$  startując od  $\gamma_0 = (0, j^*)$  używając  $T_v$ , następnie dla każdego  $i$  od 1 do  $w_p - 1$ ,  $\gamma_i = T_v(\gamma_{i-1})$ .
- Przypadek poziomy ( $w_o$  to wysokość obszaru nakładania się elementów obrazu):
  - 1) Wyliczamy skumulowany poziomy minimalny błąd  $E_h$ , gdzie  $E_h(i, w_p - 1)$ , dla  $i \in \{0, \dots, w_o - 1\}$ .

- 2) Dla każdego  $j$  od  $w_p - 2$  do 0 wyliczamy:
    - $E_h(i, j) = e(i, j) + \min(E_h(i - 1, j + 1), E_h(i, j + 1), E_h(i + 1, j + 1))$ ,  
gdzie  $j \in \{0, \dots, w_o - 1\}$ ,
    - $T_h(i, j) = \operatorname{argmin}(E_h(i - 1, j + 1), E_h(i, j + 1), E_h(i + 1, j + 1))$ .
  - 3) Definiujemy  $i^* = \operatorname{argmin}_i E_h(i, 0)$ .
  - 4) Odtwarzamy ścieżkę  $\gamma$  startując od  $\gamma_0 = (i^*, 0)$  używając  $T_h$ , następnie dla każdego  $i$  od 0 do  $w_p - 1$ ,  $\gamma_j = T_h(\gamma_{j-1})$ .
- Przypadek kształtu L:
- 1) Wyliczamy skumulowany pionowy minimalny błąd  $E_v$  oraz  $T_v$ .
  - 2) Wyliczamy skumulowany poziomy minimalny błąd  $E_h$  oraz  $T_h$ .
  - 3) Definiujemy  $i^* = \operatorname{argmin}_i (E_v(i, i) + E_h(i, i) - e(i, i))$ ,  $i \in \{0, \dots, w_o - 1\}$ .
  - 4) Odtwarzamy ścieżkę pionową  $\gamma_h$  startując od  $\gamma_h[0] = (i^*, i^*)$  używając  $T_v$ , następnie dla każdego  $i$  od 1 do  $w_p - i^* - 1$ ,  $\gamma_i = T_v(\gamma_{i-1})$ .
  - 5) Odtwarzamy ścieżkę poziomą  $\gamma_v$  startując od  $\gamma_v[w_p - 1] = (i^*, i^*)$  używając  $T_h$ , następnie dla każdego  $j$  od  $w_p - i^* - 2$  do 0,  $\gamma_j = T_h(\gamma_{j+1})$ .
  - 6) Łączymy  $\gamma_h$  z  $\gamma_v$ .

Dane wyjściowe to ścieżka, która wyznacza nam cięcie pomiędzy dwoma elementami.

## 1.5 Jądro filtra obrazu

Podczas obliczania minimalnego błędu cięcia granicznego, podjęto próbę ograniczenia przechodzenia ścieżki przez szum lub wokół jego najbliższych sąsiadów. Do uzyskania takiego rezultatu, zostało wykorzystane jądro, inaczej nazywany maską lub macierzą splotu, za pomocą którego zostały przekształcone obszary, które brano pod uwagę podczas wyznaczania ścieżki cięcia. W dalszej części pracy metoda ta została wykorzystana do rozmycia obrazu wykorzystując jądro gaussowskie przedstawione w 1.7.

**Definicja 1.9** (Dwuwymiarowy dyskretny splot [12]). Splot dyskretny na dwuwymiarowej płaszczyźnie dwóch macierzy  $\omega$  i  $f$  równy będzie

$$g(x, y) = (\omega * f)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \omega(i, j) f(x - i, y - j), \quad (1.2)$$

gdzie symbol  $[\omega * f](x, y)$  oznacza splot  $\omega$  i  $f$  w punkcie  $x, y$ .

W pracy,  $\omega$  była traktowana jako jądro filtra, a  $f$  był obszarem, który został przefiltrowany. Niech  $g(x, y)$  będzie przefiltrowanym obrazem. Każdy element jądra filtra rozważano na przedziale  $i \in [-a, a]$  oraz  $j \in [-a, a]$ . Wtedy postać splotu przybrała kształt

$$g(x, y) = (\omega * f)(x, y) = \sum_{i=-a}^a \sum_{j=-a}^a \omega(i, j) f(x - i, y - j). \quad (1.3)$$

## 1.6 Konwertowanie zapisu kolorów

Do sprawdzenia i nakładania szumów wymagano informacji o jasności piksela. Kolor piksela z obrazu był odczytany za pomocą zapisu RGB. Ten zapis nie zawierał wprost wyrażonej wartości jasności, co wynika z definicji 1.10. Dlatego zapis piksela, na który nakładano jednorodny szum, zmieniono na zapis HSL, ponieważ ma wyraźnie wyznaczoną wartość jasności wynikającą z definicji 1.11. Po zakończeniu nakładania szumu należy wykonać odwrotną operację, czyli zapis HSL zmieniono na zapis RGB.

**Definicja 1.10** (RGB). RGB to zapis koloru za pomocą odcieni czerwonego, zielonego i niebieskiego. Każdy z trzech odcieni przyjmuje liczbę całkowitą z przedziału [0, 255] [11].

**Definicja 1.11** (HSL). HSL to zapis koloru, który używa odcienia, nasycenia oraz jasności. Odcień zawiera informacje o kolorze i jego wartość całkowita mieści się w przedziale [0, 359]. Nasycenie i jasność są na przedziale [0,1] [11].

### 1.6.1 Konwersja z RGB do HSL

Poniżej pokazano przejście z zapisu RGB na zapis HSL [11]. Jak omówiono powyżej, zapis HSL był wymagany do uzyskania jasności piksela.

**Algorytm 1.12.** Dane wejściowe to odcienie  $R, G, B$  gdzie każdy kolor przyjmuje wartości całkowite w przedziale [0, 255].

1. Normalizujemy wartości RGB

$$R' = \frac{R}{255}, G' = \frac{G}{255}, B' = \frac{B}{255}.$$

2. Znajdujemy największą i najmniejszą wartość spośród  $R', G', B'$  oraz liczymy różnicę:

$$C_{\max} = \max(R', G', B'), \quad C_{\min} = \min(R', G', B'), \quad \Delta = C_{\max} - C_{\min}.$$

3. Liczymy jasność

$$L = \frac{C_{\max} + C_{\min}}{2}.$$

4. Wyliczamy nasycenie:

$$S = \begin{cases} 0, & \text{gdy } \Delta = 0, \\ \frac{\Delta}{1 - |2L - 1|}, & \text{gdy } \Delta \neq 0. \end{cases}$$

5. Wyliczamy odcień:

$$S = \begin{cases} 0, & \text{gdy } \Delta = 0, \\ 60 \cdot \left( \frac{G' - B'}{\Delta} \bmod 6 \right), & \text{gdy } C_{\max} = R', \\ 60 \cdot \left( \frac{B' - R'}{\Delta} + 2 \right), & \text{gdy } C_{\max} = G', \\ 60 \cdot \left( \frac{R' - G'}{\Delta} + 4 \right), & \text{gdy } C_{\max} = B'. \end{cases}$$

Dane wyjściowe to  $H$  na przedziale  $[0, 360)$ ,  $S$  i  $L$  na przedziale  $[0, 1]$ .

Do zamiany zapisów z RGB na HSL wykorzystano funkcję `rgb_to_hsl`, która zawarta była w bibliotece `sys_color`. Dane wejściowe dla funkcji to  $R', G', B'$ . Dane wyjściowe to  $H \in [0, 1)$ ,  $S \in [0, 1]$ ,  $L \in [0, 1]$ . Wartość całkowita  $H$  powinna być w zbiorze  $[0, 359]$ , więc została pomnożona przez 359 i zaokrąglona do liczby całkowitej.

### 1.6.2 Konwersja z HSL do RGB

Poniżej zaprezentowano metodę, która umożliwia przejście z zapisu HSL na zapis RGB [1].

**Algorytm 1.13.** Dane wejściowe to  $H, S, L$ , gdzie  $H$  przyjmuje wartości całkowite na przedziale  $[0, 359]$  oraz wartości  $S$  i  $L$  są w przedziale  $[0, 1]$ .

1. Wyliczamy chrome

$$C = (1 - |2L - 1|) \cdot S.$$

2. Szukamy punktów  $R_1, G_1, B_1$ , które będą miały takie samo nasycenie i chrome jak nasz kolor zapisany w HSL:

$$H' = \frac{H}{60},$$

$$X = C \cdot (1 - |H' \bmod 2 - 1|),$$

$$(R_1, G_1, B_1) = \begin{cases} (C, X, 0), & \text{jeśli } H' \in [0, 1), \\ (X, C, 0), & \text{jeśli } H' \in [1, 2), \\ (0, C, X), & \text{jeśli } H' \in [2, 3), \\ (0, X, C), & \text{jeśli } H' \in [3, 4), \\ (X, 0, C), & \text{jeśli } H' \in [4, 5), \\ (C, 0, X), & \text{jeśli } H' \in [5, 6). \end{cases}$$

3. By otrzymać  $R, G, B$  musimy dodać do  $R_1, G_1, B_1$  odpowiednią jasność  $m$  oraz całość pomnożyć przez 255:

$$m = L - \frac{C}{2},$$

$$(R, G, B) = (R_1 + m, G_1 + m, B_1 + m) \cdot 255.$$

Dane wyjściowe to  $R, G, B$ , gdzie wartość całkowite każdej z odcienni mieści się w przedziale  $[0, 255]$ .

Do zmiany zapisów z HSL na RGB wykorzystano funkcję `hsl_to_rgb`, która zawarta była w bibliotece `sys_color`. Dane wejściowe dla funkcji to  $H, S, L$ , gdzie każda wartość argumentu powinna być w przedziale  $[0, 1]$ . A ponieważ  $H$  była w przedziale  $[0, 359]$  1.11, więc  $H$  została znormalizowana, czyli dzielone przez 359. Dane wyjściowe z funkcji to  $R, G, B$ , gdzie wszystkie stałe były w przedziale  $[0, 1]$ . Więc stałe powinny zostać przeskalowane przez 255 i zaokrąglone do liczb całkowitych.

## 1.7 Jądro gaussowskie

Podczas próby nałożenia na piksel szumu, warto znać informacje o otoczeniu piksela. Dlatego rozmazano obraz za pomocą jądra gaussowskiego.

**Definicja 1.14** (Dwuwymiarowe jądro gaussowskie). Dwuwymiarowe jądro gaussowskie przybiera następującą postać [5, 13]

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad (1.4)$$

gdzie  $x$  i  $y$  to współrzędne,  $\sigma$  to odchylenie standardowe,  $G(x, y)$  to wartość komórki.

## 1.8 Dystrybuanty i funkcje

Do uzyskania niejednorodnego szumu użyto metodę akceptacji-odrzucenia, która bazowała na zasadzie, że im jaśniejszy piksel, tym mniejsze prawdopodobieństwo pojawienia się szumu. Podczas nakładania szumu na piksel, sprawdzano jasność piksela, a następnie przy użyciu funkcji lub dystrybuanty obliczano wartość prawdopodobieństwa, czy piksel ma być odrzucony, czy zaakceptowany. Poniżej podano kilka przykładowych dystrybuant i funkcji. Funkcja, jak i dystrybuanta, na danej dziedzinie powinny przyjmować wartości od 0 do 1.

**Definicja 1.15** (Dystrybuanta rozkładu log-normalnego). Niech  $X$  będzie z rozkładu log-normalnego

$\mathcal{LN}(\mu, \sigma^2)$ . Wtedy dystrybuanta  $F_X(x)$  rozkładu  $X$  przyjmuje postać [7]

$$F_X(x) = \Phi\left(\frac{\ln x - \mu}{\sigma}\right),$$

gdzie  $x \geq 0$ .

**Definicja 1.16** (Dystrybuanta rozkładu wykładniczego). Niech  $X$  będzie z rozkładu wykładniczego  $Exp(\lambda)$ . Wtedy dystrybuanta  $F_X(x)$  rozkładu  $X$  przyjmuje postać [4]

$$F_X(x) = 1 - e^{-\lambda x},$$

gdzie  $x \geq 0$ .

**Definicja 1.17** (Tangens hiperboliczny). Niech  $\tanh(x)$  będzie tangensem hiperbolicznym, wtedy [6]

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

W niniejszej pracy tangens hiperboliczny rozpatrywany jest na przedziale  $[0, \infty)$ , więc traktowany jest jako dystrybuanta, ponieważ spełnia warunki na dystrybuantę.

## 1.9 Działania na macierzach

W niniejszej pracy w celu uproszczenia zostały zastosowane symbole informatyczne.

**Definicja 1.18.** Niech  $M[i, j]$  to wartość komórki zlokalizowanej w macierzy  $M$  w wierszu  $i$  i kolumnie  $j$ .

**Definicja 1.19.** Niech  $M[a : b, c : d]$  to macierz rozmiaru

$$(b - a + 1) \times (d - c + 1)$$

składająca się z wartości  $M[i, j]$  dla  $i = a, \dots, b$  oraz  $j = c, \dots, d$ .

# Rozdział 2

## Implementacja

Cała implementacja została wykonana w języku Python.

### 2.1 Tworzenie jednorodnych próbek szumu

Do stworzenia jednorodnych próbek szumu został zastosowany algorytm Bridsona 1.2, który generował próbki na płaszczyźnie za pomocą próbkowania dysków Poissona. Określono rozmiar płaszczyzny ( $n \times m$ ), startowy rozmiar promienia  $r$ , początkową ilość podejść prób  $k$  do wyznaczenia nowego dysku i wyznaczanie minimalnej ilości próbek szumu, jaką program musiał wygenerować. Odpowiednią ilość szumu gwarantowała metoda relaksacji przedstawionej w podrozdziale 1.2.

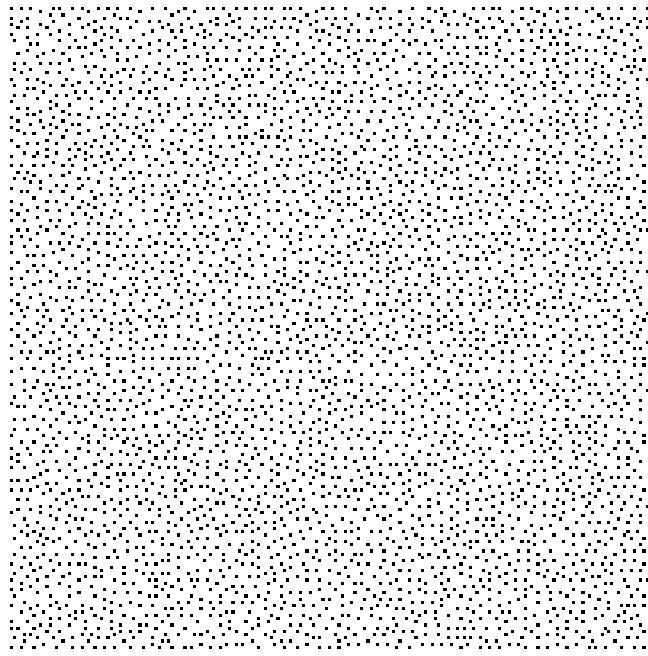
Program na początku obliczał rozmiar jednej kratki w siatce przy pomocy wzoru  $\frac{r}{\sqrt{N}}$  po czym obliczał rozmiar siatki i od razu ją tworzył. Siatkę wyrażono w postaci pustej macierzy. Współrzędne pierwszego dysku wybierano losowo z całej płaszczyzny. Następnie utworzono listę aktywnych punktów dysków, po czym dodano współrzędne centrum pierwszego dysku oraz dysk został przeniesiony do odpowiedniej komórki w siatce. Następnie program losował punkt (współrzędne dysku) z listy aktywnych punktów i względem tego punktu wylosował kąt z wartości  $[0, 360]$  oraz odległość z przedziału  $[r, 2r]$ .

Program dopasował do współrzędnych komórkę w siatce, a następnie sprawdzał, bazując na siatce, czy nowo powstały dysk miał sąsiadów w odległości dwóch kratek od kratki, gdzie miał zostać umieszczony. Jeśli tak, to sprawdzano, czy koliduje z którymkolwiek z sąsiadów. Jeśli nie, to zapisywał współrzędne dysku w komórce i dodawał centrum dysku do listy aktywnych punktów. Bazowy punkt dysku, od którego były wyznaczane nowe punkty dysków, także zostawał w liście. Jeśli pojawiła się kolizja, podjęto kolejną próbę umieszczenia nowego dysku na płaszczyźnie. Jeśli po  $k$  próbach nie udało się umieścić żadnego nowego dysku, bazowy punkt dysku, od którego wyznaczane były nowe dyski, był usuwany z listy aktywnych punktów. Cały proces powtarzano, dopóki lista aktywnych punktów nie była równa zeru. Na koniec program zwracał listę współrzędnych centrów dysków, które były próbami szumu.

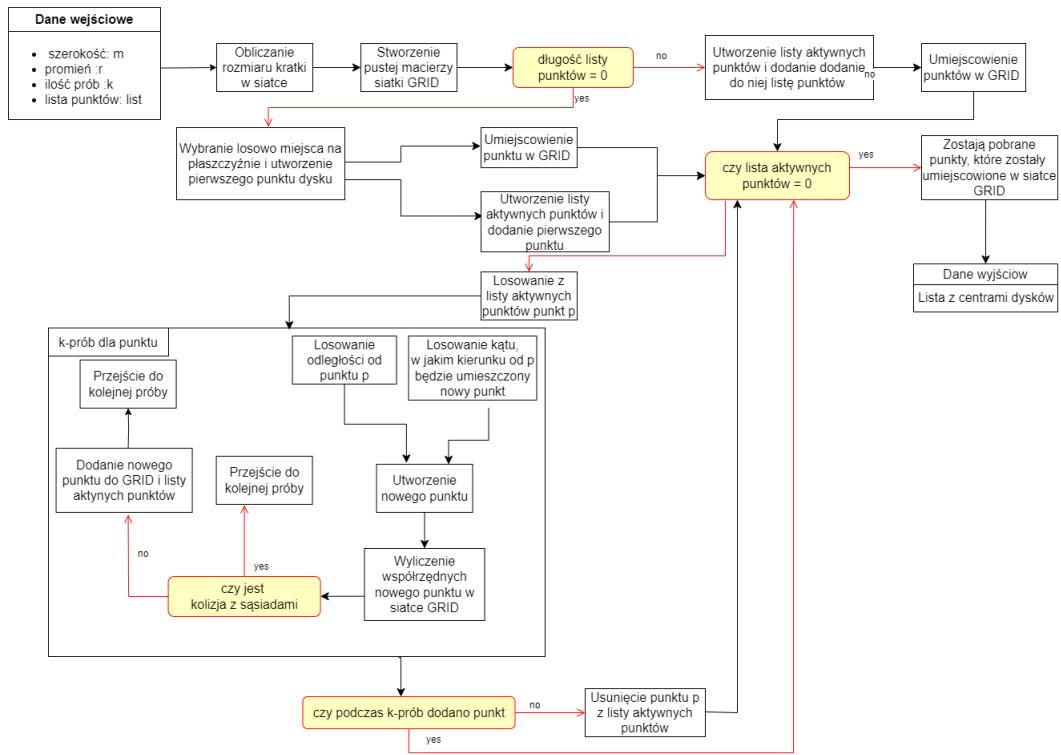
Jeśli ilość próbek była mniejsza od zakładanej, używano metody relaksacji, która zmniejszała  $r$  za pomocą mnożenia przez 0,9 oraz zwiększała ilość prób  $k$  o jeden. Następnie ponownie wykorzystano algorytm Bridsona ze zmienionym wartościami, a początkowa lista aktywnych punktów była równa punktom, które zostały zaakceptowane dla poprzedniego  $r$ .

Cały proces generowania jednorodnego szumu zakończano gdy ilość próbek szumu była równa lub większa od żądanej na początku wartości. Współrzędne były przenoszone z listy na macierz binarną, gdzie 1 to szum a 0 to brak szumu. Wizualizację implementacji

algorytmu Bridsona przedstawiono na diagramie 2.2.



Rysunek 2.1: Szum uzyskany z próbkowania dysków Poissona.



Rysunek 2.2: Diagram do algorytmu Bridsona.

## 2.2 Tworzenie kafelków szumu

Do stworzenia zestawu kafelków potrzebne były cztery segmenty o rozmiarach  $n \times n$  z przestrzeni wypełnionej szumami, które następnie były oznaczane jako próbki szumu.

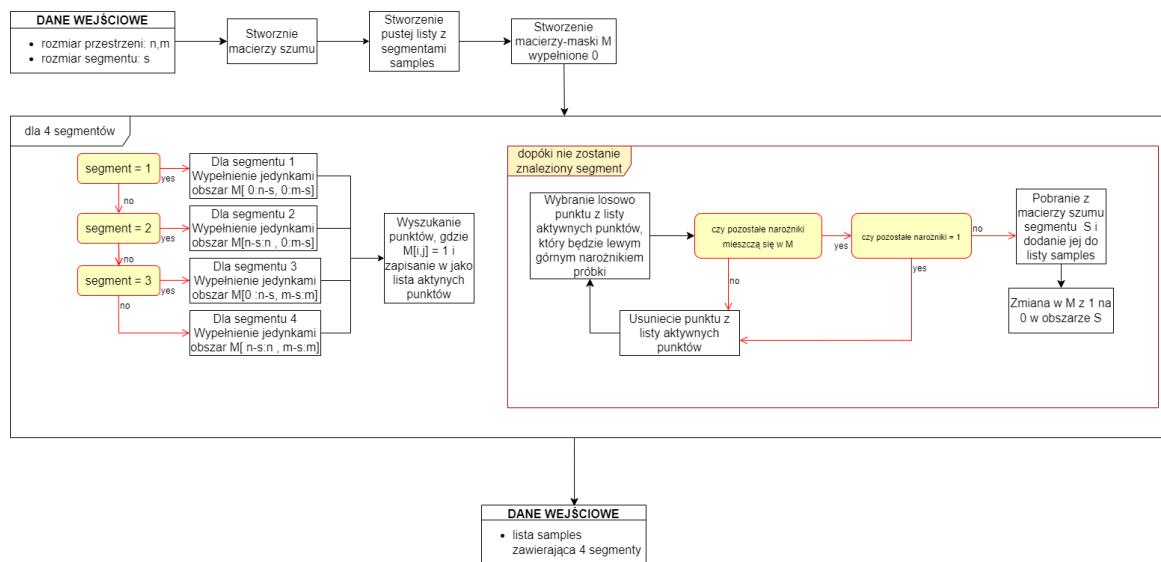
Do próbek były przyporządkowane kolory (na przykład żółty, czerwony, zielony, niebieski). Zdefiniowano zestaw kafelków, ustawiając kolory ścian pionowych i poziomych (na przykład [czerwony, zielony] dla ścian pionowych i [żółty, niebieski] dla ścian poziomych), po czym tworzono kafelki, używając jądra oraz minimalnego błędu cięcia granicznego. Pod koniec zrotowano kafelki o  $45^\circ$  i z tak powstałych rombów wycięto odpowiednie kwadraty, którymi były kafelkami.

## 2.2.1 Pobieranie segmentów szumu z przestrzeni

Segmenty pobrano z macierzy szumu  $N$  o wymiarze  $(n \times m)$ , której generacja została pokazana w podrozdziale 2.1. Program oprócz macierzy potrzebował informację o rozmiarze segmentu  $(s \times s)$ . Proces zaczynał się od stworzenia macierz-maski  $M$  wypełnionej zerami. Dla pierwszego segmentu wyznaczano obszar, w którym mógł się poruszać podczas szukania odpowiedniego miejsca. Obszar ten ukazano za pomocą jedynek w masce  $M$ . Wartości komórki w  $M[0 : n - s, 0 : m - s]$  była równa jeden. Pobierano współrzędne komórek, w których ulokowane zostały jedynki i za ich pomocą utworzono listę aktywnych punktów.

Program losował z listy aktywnych punktów punkt, który odpowiadał lewemu górnemu narożnikowi i sprawdzano pozostałe narożniki, czy wartość ich była równa jeden. Jeśli wszystkie pozostałe narożniki były równe jeden, wtedy program akceptował wybrany punkt, pobierając segment z macierzy  $N$ , gdzie lewy górny narożnik segmentu równy był współrzędnym badanego punktu oraz w masce na takim samym obszarze zamieniano wartości komórek z jedynek na zera. W przeciwnym wypadku lub gdy pozostałe narożniki wychodziły poza maskę, punkt był usuwany z listy aktywnych punktów. Punkty były losowane, dopóki nie został odnaleziony segment (rozmiar obszaru gwarantuje znalezienie segmentu).

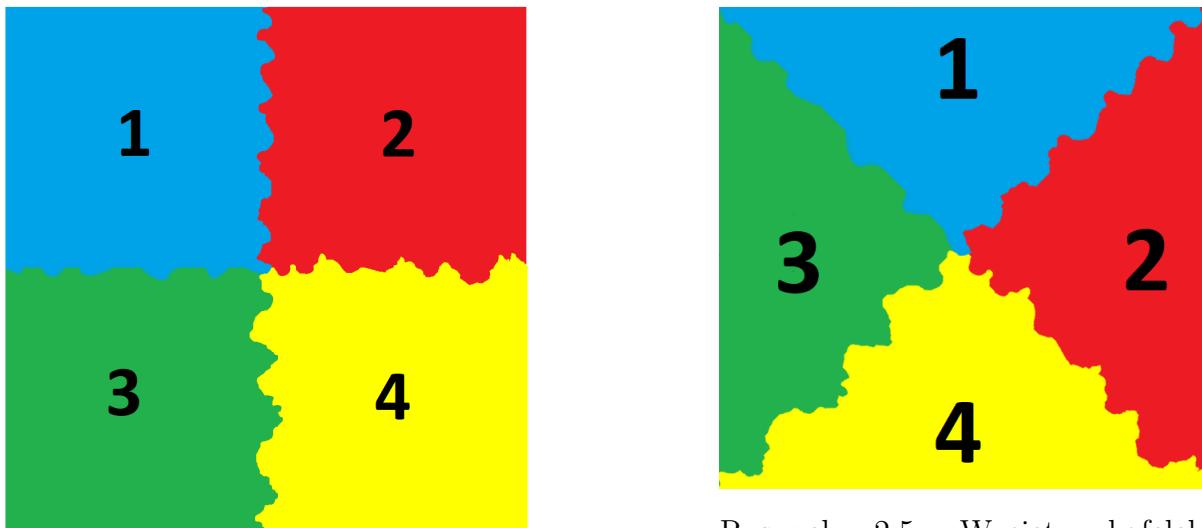
Dla segmentu numer dwa modyfikowano maskę  $M[n - s : n, 0 : m - s]$ , zastępując jedynkami zera. Dla trzeciego segmentu ta sama modyfikacja była stosowana dla  $M[0 : n - s, m - s : m]$ . Modyfikacja maski dla ostatniego segmentu miała miejsce w  $M[n - s : n, m - s : m]$ . Program zwracał cztery segmenty, które zostały wybrane losowo i nie pokrywały się. Wizualizację implementacji przedstawiono na diagramie 2.3.



Rysunek 2.3: Diagram do pobierania segmentów.

### 2.2.2 Tworzenie pełnego zestawu kolorów kafelków

Po wyznaczeniu czterech próbek szumu zdefiniowano zestawu kolorów. Kolory przyporządkowano każdej próbce. Zdefiniowanie zestawu oznaczało wyznaczenie kolorów, które mogłyby występować dla danego rodzaju ściany (w tym przypadku był to rodzaj pionowy i poziomy). Program zwracał listę kombinacji ścian kafelków. Każdy element listy składał się z kolorów ścian oraz odpowiednio do nich przypisanych próbek. Ostatecznie otrzymano listę, w której element składał się z: [górną ścianą, prawa ścianą, lewa ścianą, dolna ścianą]



Rysunek 2.4: Stworzenie kafelka bez rotacji.

Rysunek 2.5: Wycięty kafelek z rombu, który był obrazem 2.4 obróconym o  $45^\circ$ .



Rysunek 2.6: Wybrane losowo kafelki z zestawu.

### 2.2.3 Tworzenie kafelka

Przed łączeniem kafelków zdefiniowano jądro  $F$ , za pomocą którego były modyfikowane próbki szumu. Próbki te zostały połączone za pomocą metod minimalnego błędu cięcia granicznego. Jądro filtra  $F$  w tej pracy wynosiło:

$$\begin{bmatrix} 0,25 & 0,25 & 0,25 & 0,25 & 0,25 \\ 0,25 & 0,5 & 0,5 & 0,5 & 0,25 \\ 0,25 & 0,5 & 1 & 0,5 & 0,25 \\ 0,25 & 0,5 & 0,5 & 0,5 & 0,25 \\ 0,25 & 0,25 & 0,25 & 0,25 & 0,25 \end{bmatrix} \quad (2.1)$$

Po zdefiniowaniu  $F$  rozpoczęto łączenie próbek. Wyznaczono jaki obszar o dwóch próbkach powinien nakładać się na siebie. W programie była to wartość całkowita działania  $\frac{s}{6}$ . Następnie pobierano próbki dla pierwszego kafelka z zestawu, który miał być tworzony.

Pobierano pierwszą próbke z listy (w programi pierwsza próbka odpowiadała górnej ścianie, druga próbka to prawa ściana kafelka, trzecia to lewa ściana a ostatnia czwarta to dolna ściana). Utworzono macierz  $Out$  o wymiarach  $2s - o \times 2s - o$ , na której zapisywano wyniki łączenia próbek. Pierwsza próbka bez żadnych zmian została umieszczona w lewym górnym rogu macierzy  $Out$ .

Następnie pobierano drugą próbke iłączono z pierwszą próbką, która była zapisana w macierzy  $Out$ , za pomocą metody minimalnego błędu cięcia granicznego. Na wstępnie próbki były obrabiane za pomocą filtra  $F$ . Bazując na algorytmie 1.4 wyznaczano najlepszą ścieżkę, która była oznaczona jako  $\gamma$ . Na początku utworzono pustą macierz  $E$  do zapisania kosztu budowy poszczególnych długości ścieżek i oraz macierz  $T$  do przechowania informacji, który z trzech dolnych sąsiadów posiadał najmniejszą wartość. Wycinano obszary próbek, oznaczone jako  $B^1$  i  $B^2$ , które nachodziły na siebie, po czym obliczono  $e$ , które było równe  $B^1 + B^2$ . Dolny rzad macierzy  $E$  zastępowano dolnym rzędem macierzy  $e$ . Startując od  $s - 2$  do 0 znajdowano minimalną wartość z trzech dolnych sąsiadów  $E[i, j]$ , czyli  $E[i + 1, j - 1]$ ,  $E[i + 1, j]$  oraz  $E[i + 1, j + 1]$ , po czym do rezultatu dodawano wartość komórki  $e[i, j]$  i w ten sposób otrzymywano wartość komórki  $E[i, j]$ . Dla  $i = 0$  minimum było wyszukiwane tylko w  $E[i + 1, j]$  i  $E[i + 1, j + 1]$ , a dla  $i = o - 1$  minimum wyszukiwano tylko w  $E[i + 1, j - 1]$  i  $E[i + 1, j]$ . Jednocześnie wyznaczano wartość  $T[i, j]$ . Wyszukano komórkę w pierwszym rzędzie  $E$  z najmniejszą wartością a jej współrzędne zapisano na początku  $\gamma$ . Zaczynając od drugiego elementu, kolejne miejsca biegienia ścieżki  $\gamma$  wyznaczano poprzez rekurencję w postaci  $\gamma[i] = T[\gamma[i - 1]]$ , gdyż musi zostać wybrany sąsiad  $\gamma[i]$ . Po uzyskaniu ścieżki  $\gamma$  program przechodził do łączenia dwóch próbek.

Z dwóch próbek pobierano obszary  $o \times s$ , które nakładały się wzajemnie i tworzono maskę  $M$  o rozmiarach  $o \times s$  z samymi zerami. Współrzędne punktów ze ścieżki  $\gamma$  nakładano na  $M$ , zastępując zera jedynkami. Na lewej części  $M$ , którą wyznaczona była przez  $\gamma$ , zastępowano zera jedynkami. Przygotowaną maskę  $M$  nakładano na obszar pobrany z macierzy  $Out$ , natomiast na obszar pobrany z drugiej próbki nałożono  $1 - M$ . W ten sposób uzyskano połączenie dwóch nakładających się obszarów. Następnie połączono próbki ze sobą. Próbka druga zostaje zmodyfikowana w obszarze nakładania, a następnie umieszczono ją w  $Out[0 : s, s - o : 2s - o]$ , gdzie jednocześnie nadpisywano obszar nachodzenia w próbce pierwszej.

Próbkę trzecią łączono z próbką, która była zlokalizowana w  $Out[0 : s, s - o : 2s - o]$ . Miejsce obszarów nakładania zmienił się z pionu na poziom, a więc z próbki pobranej z  $Out$  pobrano obszar z dolnej jej części, a z próbki trzeciej pobrano obszar z górnej jej części. Proces poruszał się po kolumnach wyznaczającą ścieżkę. W masce  $M$  zmieniał się rozmiar na  $(o \times s)$  oraz ścieżka przedzielała  $M$  na część górną i dolną. Ostatecznie zmodyfikowana próbka trzecia była dodana do macierzy  $Out[s - o : 2s - o, 0 : s]$ .

Podczas dołączania próbki czwartej do macierzy  $Out$ , korzystano z cięcia typu  $L$  przedstawionego w algorytmie 1.4. Próbka góra to  $Out[0 : s, s - o : 2s - o] = P_{oldh}$ , natomiast próbka lewa to  $Out[s - o : 2s - o, 0 : s] = P_{oldv}$ . Program modyfikował próbki za pomocą jądra  $F$ , po czym rozpoczęło się łączenie próbek za pomocą minimalnego błędu cięcia granicznego. Niech  $E_h$ ,  $T_h$  i  $e_h$  to macierze  $E$ ,  $T$ ,  $e$  wyznaczone dla przypadku łączenia poziomego oraz  $E_v$ ,  $T_v$  i  $e_v$  to macierze  $E$ ,  $T$ ,  $e$  wyznaczone dla łączenia pionowego. Po otrzymaniu wszystkich macierzy przechodzono do znalezienia punktu startowego dla ścieżki, a raczej dwóch ścieżek, które powinny być wyliczone i połączone.  $P_{oldh}$  i  $P_{oldv}$  posiadały jeden wspólny kawałek, który znajdował się w obydwóch obszarach nakładania się. Jego wymiar to  $o \times o$ , w którym zlokalizowany był punkt startowy. Proces zaczynał

się od znalezienia punktu początkowego, który wyliczano ze wzoru

$$i^* = \operatorname{argmin}_i (E_v[i, i] + E_h[i, i] - e[i, i]), \quad i = 0, 1, \dots, o - 1. \quad (2.2)$$

Utworzono dwie ścieżki. Jedna dla łączenia poziomego  $\gamma_h$ , druga dla pionowego  $\gamma_v$ . Dla ścieżki  $\gamma_h$ , ścieżkę wyznaczano od końca za pomocą  $T_h$ , tak, że

$$\gamma_h[j] = T_h[\gamma_h[j - 1]],$$

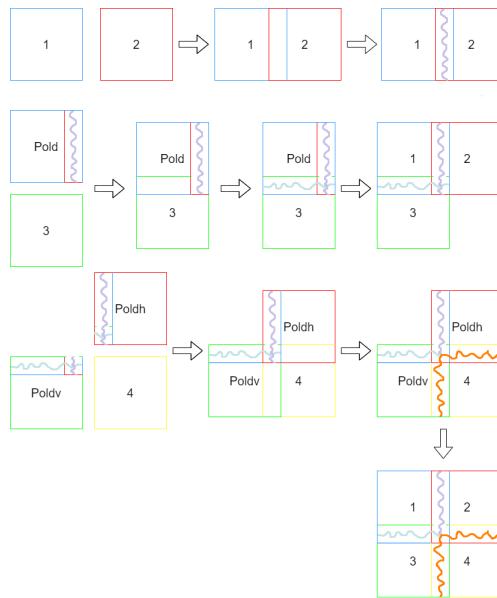
gdzie  $\gamma_h[0] = (i^*, i^*)$ . Przechodząc przez punkt startowy  $i^*$ ,  $i^*$  rozpoczynano liczenie ścieżki dla  $\gamma_h$ , w sposób następujący

$$\gamma_v[i] = T_v[\gamma_v[i + 1]],$$

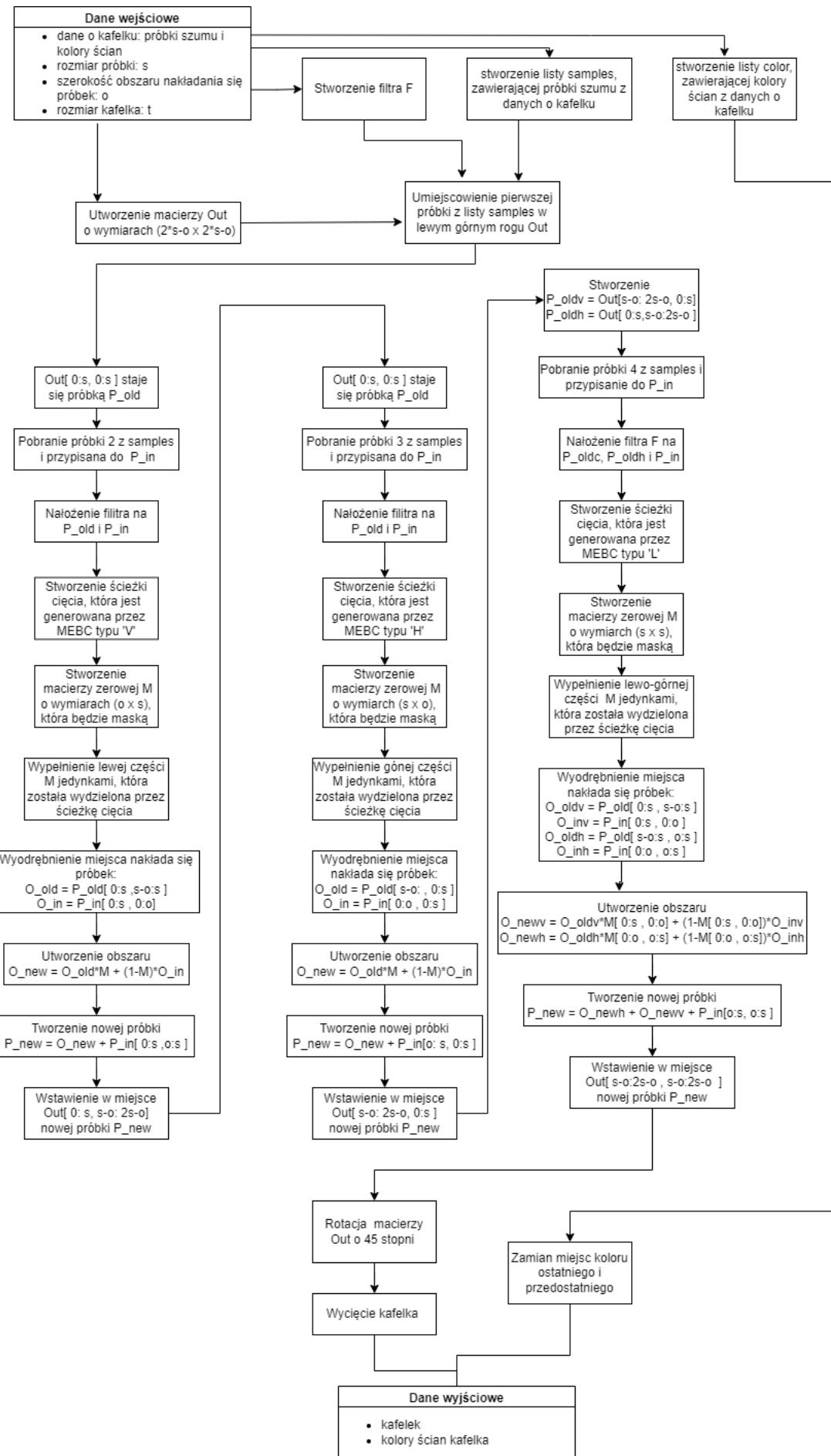
gdzie  $\gamma_h[s - 1] = (i^*, i^*)$ . Na koniec, ścieżki  $\gamma_h$  i  $\gamma_v$  zostały połączone, czego rezultatem jest ścieżka  $\gamma$ .

Została stworzona maska  $M$ , w którym to przypadku miała rozmiar  $s \times s$ . Kolejno na  $M$  została nałożona ścieżka, zastępując zera jedynkami oraz wypełniając lewą i górną część  $M$  jedynkami. Następnie z maski wycinano obszar łączenia pionowego  $M[0 : s, 0 : o] = M_v$  oraz niepełen kawałek z odcinka poziomego  $M[0 : o, o : s] = M_h$ . Nakładano maskę  $M_h$  na obszary  $P_{oldh}[s - o : s, o : s]$ , a  $1 - M_h$  nałożono obszar nakładana próbki czwartej  $[0 : o, o : s]$ . Po czym na obszar  $P_{oldv}[0 : s, s - o : s]$  nakładano  $M_v$ , oraz na obszar czwartej próbki  $[0 : s, 0 : o]$  nałożono  $1 - M_v$ . Na koniec w  $Out[s - o : 2s - o, s - o : 2s - o]$  umieszczało zmodyfikowaną próbke czwartą.

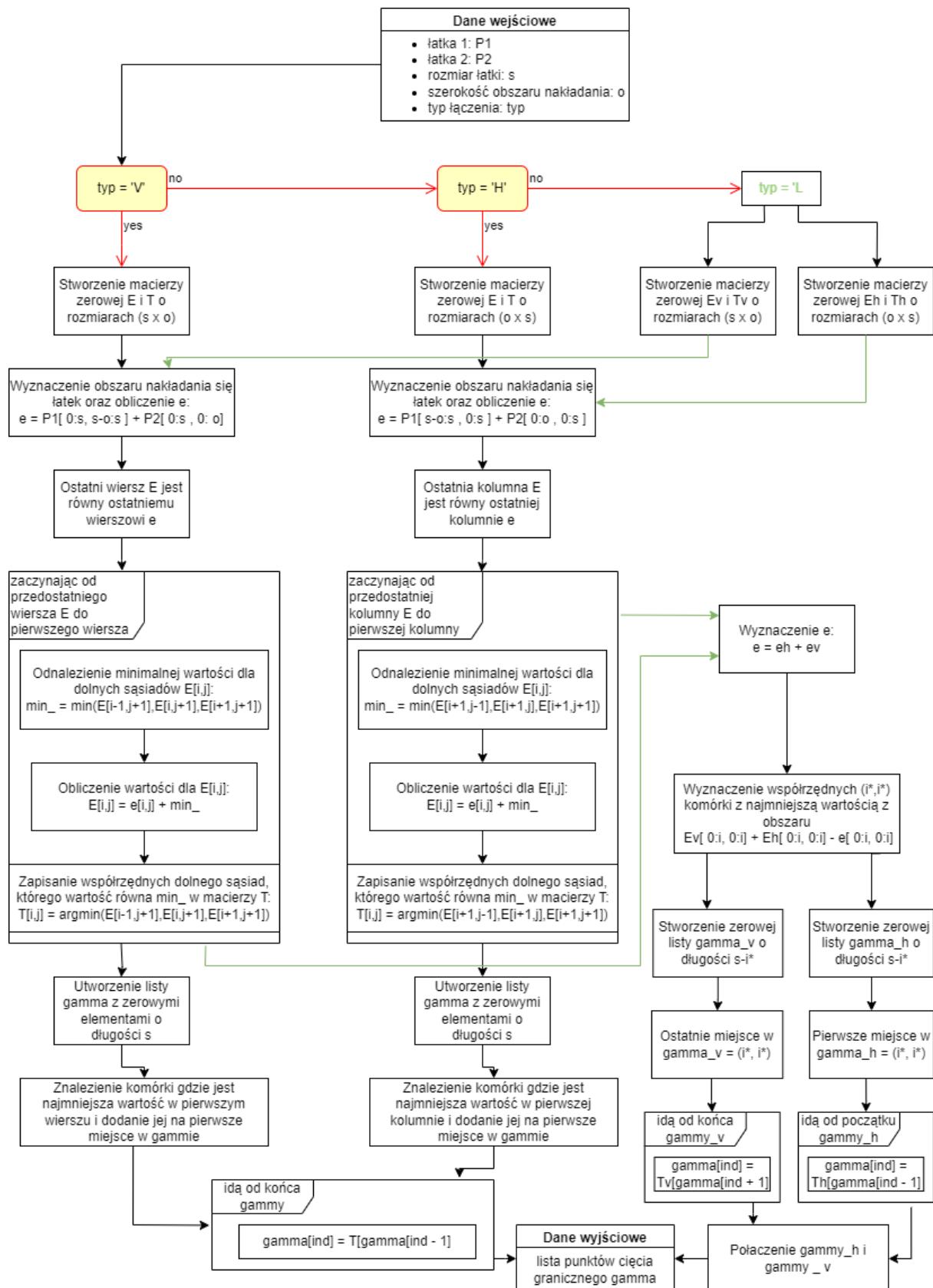
Po zakończeniu procesu otrzymywano kafelek o wizualizacji (rysunek 1.3). Obrócono więc kafelek, oznaczony jako  $RT$ , o kąt  $45^\circ$ , a następnie przycięto go. W programie tę czynność wykonywała funkcja `rotate` z parametrem `reshape = True`, by powyższy kafelek nie był modyfikowany podczas obracania. Wyznaczano rozmiar  $RT$ , oznaczony jako  $x$ , i pobrano liczbę całkowitą  $\lfloor \frac{x}{2} \rfloor$ , która wyznaczała środek obróconego kafelka. Wyliczano połowę długości  $t$  ściany kafelka  $\frac{t}{2} = \emptyset$ , po czym z obróconego kafelka wycięto obszar  $RT[x - \emptyset : x + \emptyset, x - \emptyset : x + \emptyset]$ . Na koniec w liście kolorów ścian, zamieniono miejscami ostatni koloru ściany z przedostatnim kolorem ściany, z wymogów do funkcji układającej parkietaż. Wizualizacja kafelka została zaprezentowana poglądowo na rysunkach 2.4–2.7. Wizualizację implementacji przedstawiono na diagramach 2.8 i 2.9.



Rysunek 2.7: Wizualne przedstawienie procesu tworzenia kafelka.



Rysunek 2.8: Diagram pokazujący działanie programu do tworzenia kafelków.



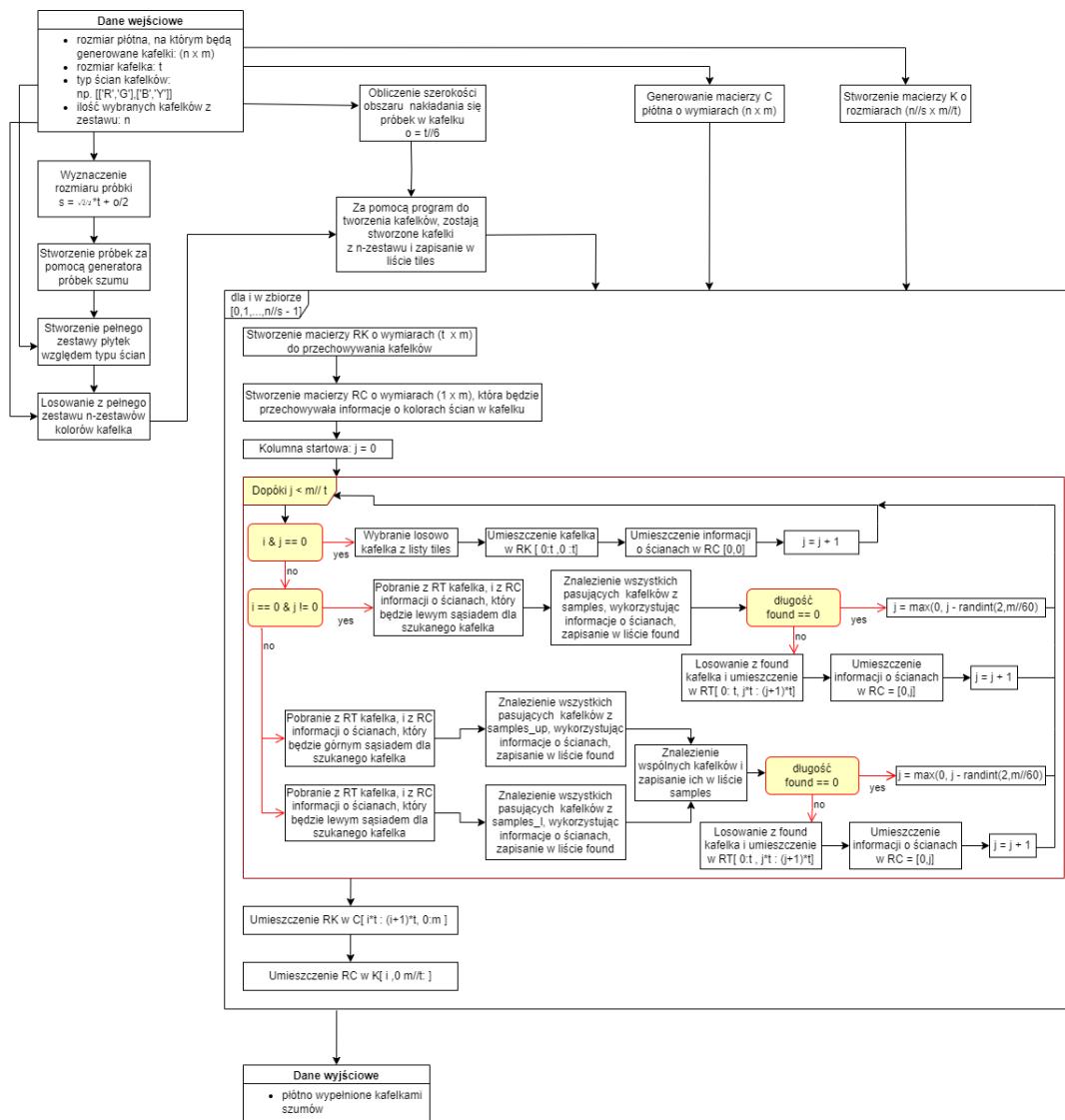
Rysunek 2.9: Diagram pokazujący program do znajdowania ścieżki dla minimalnego błędu cięcia granicznego.

## 2.3 Generowanie jednorodnego szumu obrazu

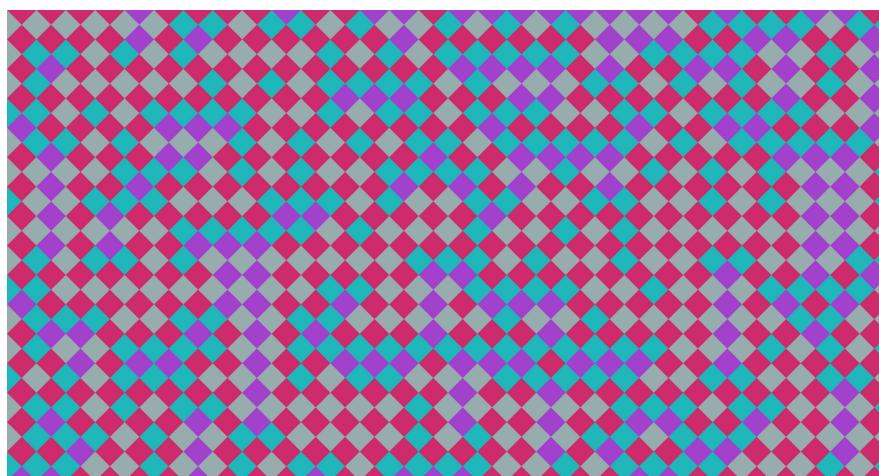
W celu wygenerowania jednorodnego szumu zostały wykorzystane stochastyczne kafelki Wanga 1.3. Zdefiniowano rozmiary płótna  $n \times m$ , na które były nakładane kafelki oraz rozmiar kafelka  $t$ . Płótno było macierzą  $C$  wypełnioną zerami. Została utworzona druga pusta macierz  $K$  o wymiarach  $\lfloor \frac{n}{t} \rfloor \times \lfloor \frac{m}{t} \rfloor$ , która przechowywała informacje o kolorach ścian danego kafelka. Następnie został zdefiniowany obszar nakładania się próbek podczas tworzenia kafelka w procesie zaprezentowanym w podrozdziale 2.2.3. Obliczono rozmiar próbki  $s = \frac{\sqrt{2}}{2}t + \frac{\sigma}{2}$ , którą pobierano z przestrzeni wygenerowanego bazowego szumu w podrozdziale 2.2.1 i pełen zestaw koloru kafelków pokazanych w podrozdziale 2.2.2. Program losował z pełnego zestawu żądaną  $n$  ilość kolorów kafelków, który oznaczono jako  $n - zestaw$ , po czym tworzył z nich kafelki (zbyt mała ilość kafelków względem pełnego zbioru powodowała bardzo długi proces renderowania płaszczyzny lub nawet prowadziła do niepowodzenia).

Dla każdego wiersza  $i$  z macierzy  $K$  tworzono pustą macierz  $RC$  o wymiarach  $(1 \times \lfloor \frac{m}{t} \rfloor)$ , która odpowiadała za informacje o kolorach ścian kafelka, oraz macierz  $RT$  o wymiarach  $(t \times m)$ , która odpowiadała za przechowywanie kafelków. Dla pierwszego wiersza  $K$  pierwszy kafelek został wylosowany z  $n - zestawu$  i zapisany w  $RT$ , a dane o jego ścianach w  $RC$ . Sprawdzano kolor prawej ściany lewego sąsiada szukanego kafelka, wykorzystując macierz  $RC$ , i dobierano kandydatów, których kolor lewej ściany pasował do lewego sąsiada. Z kandydatów losowano kafelek, który wkleiano na miejsce szukanego kafelka do macierzy  $RT$ . Niech  $j$  oznaczał bieżącą kolumnę. Jeśli nie dopasowano żadnego kafelka, wówczas program cofał się o  $\max(0, j - \text{randint}(2, \lfloor \frac{m}{t} \rfloor \cdot 0,1))$  kafelków. Po uzupełnieniu macierzy  $RT$  i  $RC$  macierze te wkleiano odpowiednio do  $C[i \cdot t : (i + 1) \cdot t, 0 : m]$  oraz  $RC$  do  $K[i, 0 : \lfloor \frac{m}{t} \rfloor]$ .

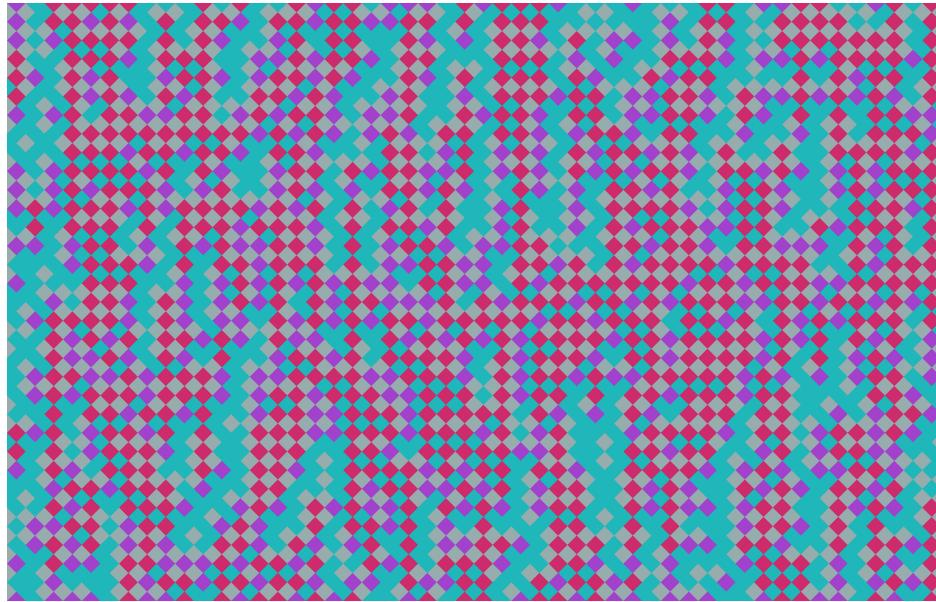
Dla wszystkich kolejnych wierszy działania były analogicznie. Program dobierał pierwszy kafelek, sprawdzając dolny kolor ściany górnego sąsiada, przy pomocy  $K$ , kolejno dopasowywał kandydatów, by kolor ich górnej ściany zgadzał się z kolorem dolnej ściany górnego kafelka. Losowo wybierał z kandydatów kafelek i umieszczał go się w macierzach  $RT$  i  $RC$ . Dla kolejnych szukanych kafelków program sprawdzał prawy kolor ściany lewego sąsiada z macierzy  $RC$  oraz osobno dolny kolor ściany górnego sąsiada z macierzy  $K$ . Program porównywał uzyskane kafelki z dwóch wyszukań i wybierał wspólne kafelki. Z wybranych kafelków losował jeden kafelek i umieszczał go w  $RT$  i  $RC$ . Jeżeli program nie znajdował żadnego odpowiedniego kafelka, cofał się tak jak w przypadku pierwszego wiersza. Gdy wypełniono macierze  $RT$  i  $RC$ , zostawały odpowiednio dodawane do macierzy  $C$  i  $K$ . Wizualizację implementacji przedstawiono na diagramie 2.10, a wypełnione płótno kolorowymi kafelkami Wanga ukazano na rysunkach 2.11 i 2.12



Rysunek 2.10: Diagram przedstawiający implementacje kafelków Wanga.



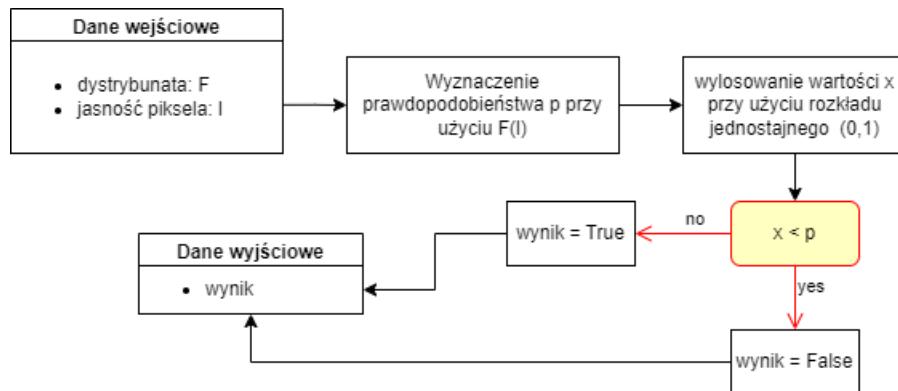
Rysunek 2.11: Tło wypełnione stochastycznymi kafelkami Wanga z zestawu przedstawionego na rysunku 2.6.



Rysunek 2.12: Tło wypełnione stochastycznymi kafelkami Wanga z zestawu  $[R,B,G] \times [Y,B]$ .

## 2.4 Tworzenie niejednorodnego szumu

Program do stworzenia niejednorodnego szumu wymagał informacji o jasności piksela z rozmytego obrazu oraz dystrybuantę, która zwracała wartość prawdopodobieństwa  $p$  nałożenia szumu na piksel. Po uzyskaniu  $p$  sprawdzano, czy szum dla piksela został zaakceptowany. Program wyliczał  $p$  podkładając jasność piksela pod dystrybuantę, po czym losował liczbę z rozkładu  $\mathcal{U}(0,1)$  i sprawdzał, czy jest ona mniejsza od prawdopodobieństwa  $p$ . Jeśli tak program zwracał False, a więc szum był odrzucony. W przeciwnym wypadku szum został zaakceptowany i zwracał True. Wizualizację implementacji przedstawiono na diagramie 2.13.



Rysunek 2.13: Diagram przedstawiający implementację metodę akceptacji-odrzucenia.

## 2.5 Nakładanie szumu na obraz

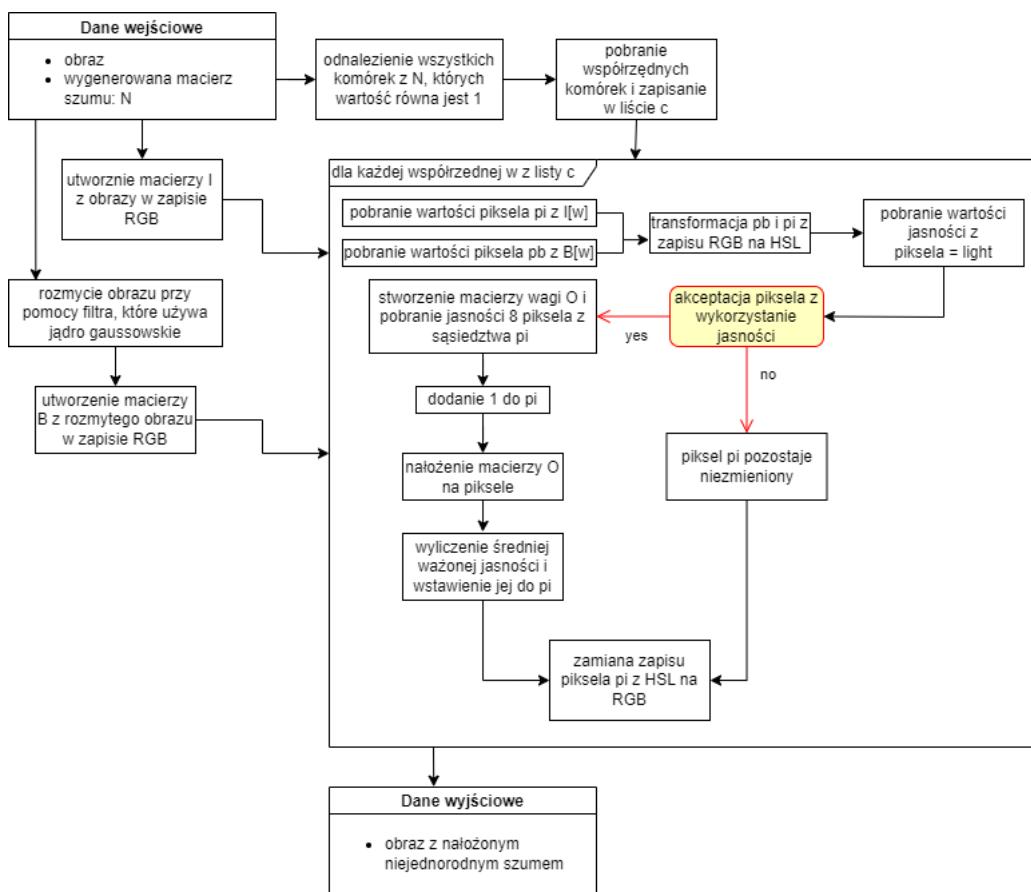
Wprowadzano obraz oraz jego zapis macierzowy  $I$ , którego piksele były wyrażone w zapisie RGB, oraz macierz binarną jednorodnego szumu wygenerowanego przy pomocy próbkowania dysków Poissona, oraz parkietażu zaprezentowanego w podrozdziale 2.3. Obraz został

rozmazany przy pomocy jądra gaussowskiego, który był omawiany w podrozdziale 1.7 za pomocą funkcji `ImageFilter.GaussianBlur`, która była zawarta w bibliotece `Pillow`. Obrazy był rozmazywany, ponieważ bez dodatkowych obliczeń można było dostać informacje o sąsiadach piksela.

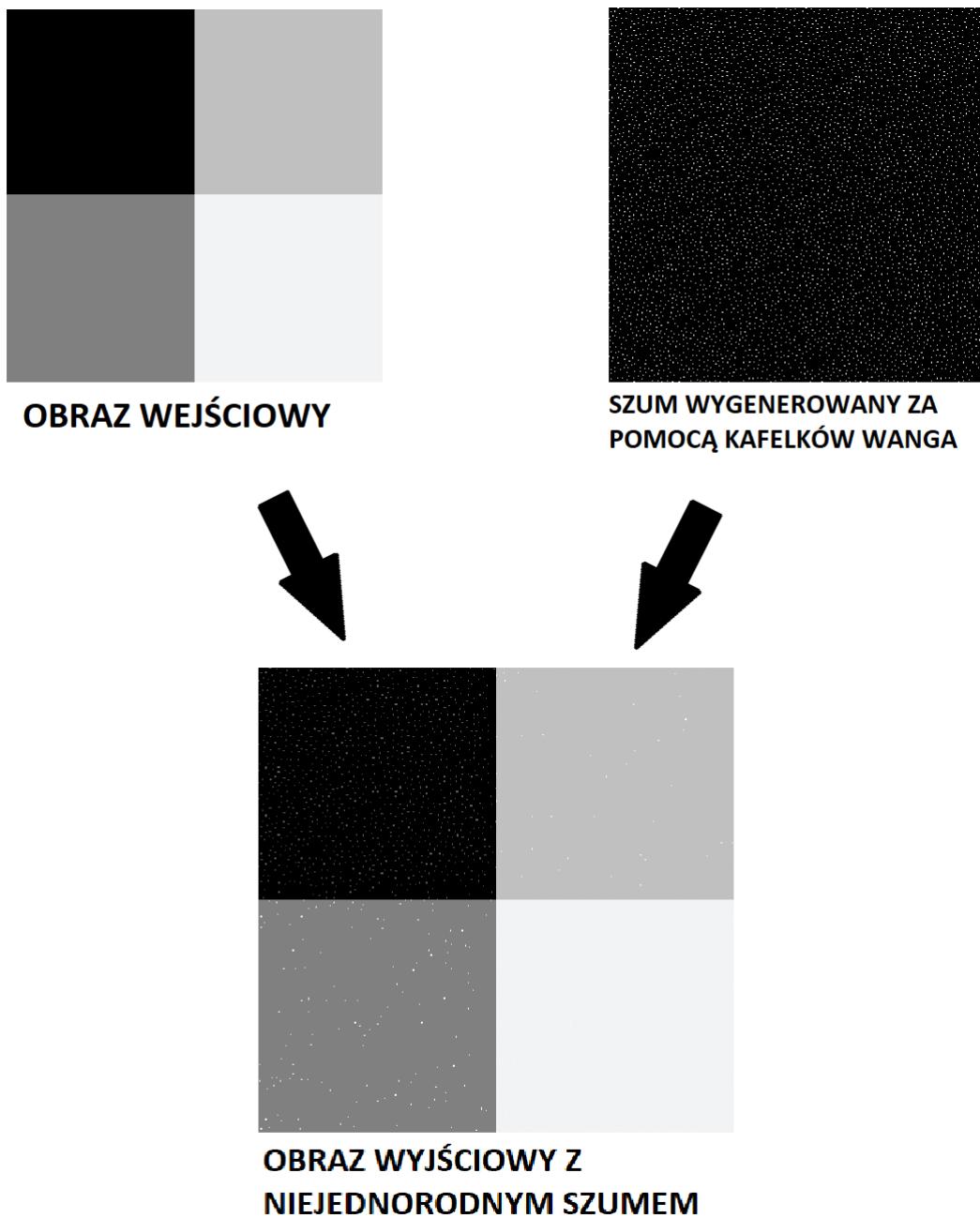
Rozmyty obraz zapisywano w macierzy  $B$ , gdzie piksele również były w formacie RGB. Współrzędne komórek, których wartość wynosi 1, pobrano z macierzy szumu i zapisano w liście  $c$ , po czym na każdym elemencie  $w$  z listy  $c$  zostały wykonane podane poniżej działania. Pobrano piksel z obrazu  $I$  o współrzędnych  $w$  i analogicznie piksel z rozmytego obrazu  $B$ . Zapis pikseli przetransformowano na HSL przy pomocy metody zaprezentowanej w podrozdziale 1.6.1. Za pomocą programu pokazanego w podrozdziale 2.4, którego argumentem była jasność, która jest trzecim elementem w pikselu w zapisie HSL, oraz dystrybuanta, sprawdzano, czy piksel został zaakceptowany.

Jeśli piksel został zaakceptowany, to jasność piksela została wyliczona poprzez sąsiadów piksela oraz macierzy wagi  $O$ . Z obrazu  $I$  pobierano 8 pikseli sąsiadujących z pikselem, na który miał być nałożony szum. Za pomocą  $O$  ustalano wagi dla poszczególnych jasności sąsiadów piksela, po czym sumowano wszystkie piksele, do sumy dodawano 1. Wynik dzielono przez sumę wartości  $O$  i w ten sposób otrzymywano jasność dla piksela. W przypadku dużego rozmiaru obrazu zmiana jasności rozszerzała się na sąsiadów piksela. W momencie odrzucenia jasność piksela zostawała niezmieniona.

Piksel sformatowano z powrotem do postaci RGB, przy pomocy 1.6.2, i zapisywano w  $I[w]$ . Po sprawdzeniu wszystkich elementów w liście  $c$  program zakończył się i zwracał macierz obrazu  $I$  z nałożonym niejednorodnym szumem. Wizualizację implementacji przedstawiono na diagramie 2.14.



Rysunek 2.14: Diagram przedstawiający proces nakładania niejednorodnego szumu na obraz.

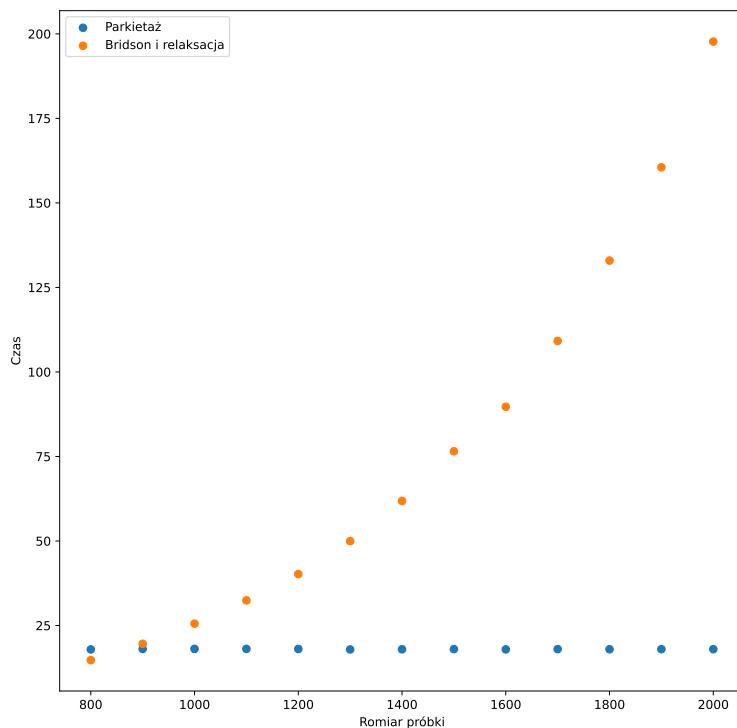


Rysunek 2.15: Pokazowe działanie programu. Użyta dystrybuanta to  $Exp(\lambda = 1)$ .

## Rozdział 3

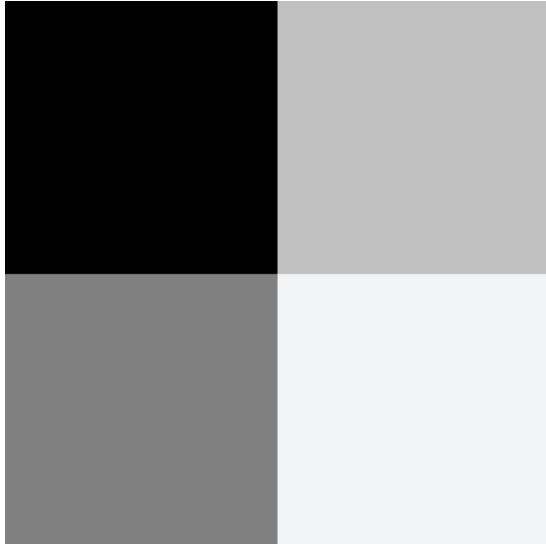
### Wyniki własne

W tym rozdziale zaprezentowano wyniki implementacji autorskiego algorytmu. Z wykorzystaniem metody Monte Carlo, wyliczono średni czas generowania jednorodnego szumu, używając tylko algorytmu Bridsona oraz metody relaksacji na płaszczyzny o wymiarach  $n \times n$ , gdzie  $n = 800, 900, \dots, 2000$ . Za pomocą metody Monte Carlo został wyliczony średni czas generowania jednorodnego szumu wykorzystując parkietaż dla takich samych płaszczyzn. Kafelki do parkietażu mają rozmiar  $100 \times 100$ . Na koniec nałożono czasy dwóch sposobów na wykres i uzyskano wykres 3.1.

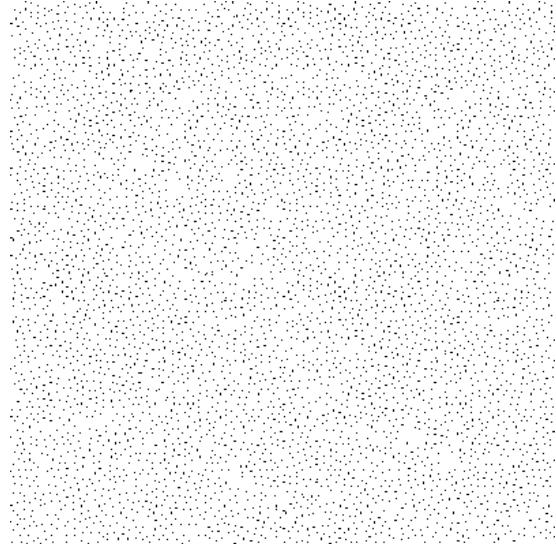


Rysunek 3.1: Wykres przedstawiający czas generowania szumów względem rozmiaru płaszczyzny próbki dla dwóch różnych sposobów.

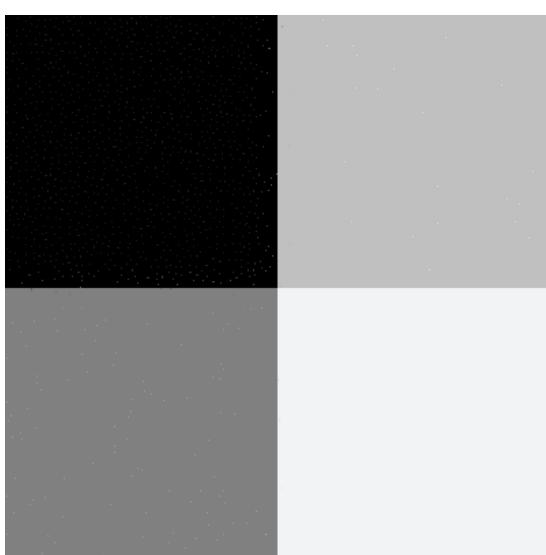
Na przedstawionym wykresie 3.1 zależności czasu generowania jednorodnego szumu względem rozmiaru płaszczyzny dla algorytmu Bridsona i metody relaksacji czas generowania rośnie wykładniczo względem rozmiarów płaszczyzny. Natomiast czas generowania jednorodnego szumu dla parkietu, czyli stochastycznych kafelków Wanga, jest o wiele krótszy niż dla pierwszego sposobu oraz stosunkowo stały.



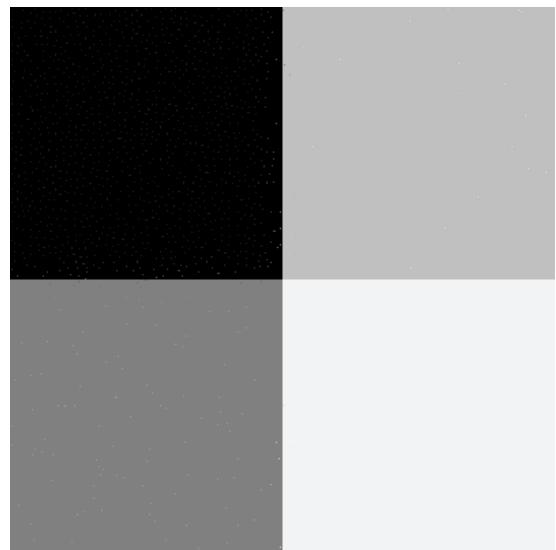
Rysunek 3.2: Oryginalne obraz.



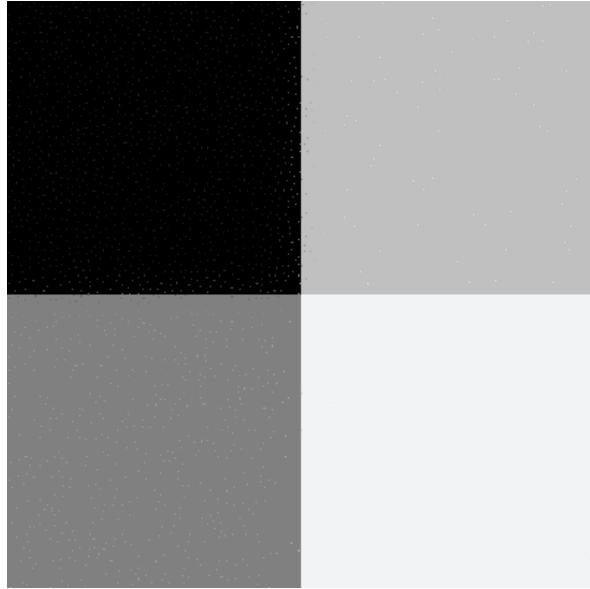
Rysunek 3.3: Jednorodny szum.



Rysunek 3.4: Użyta funkcja do metody akceptacji-odrzucenia to  $\tanh$ .



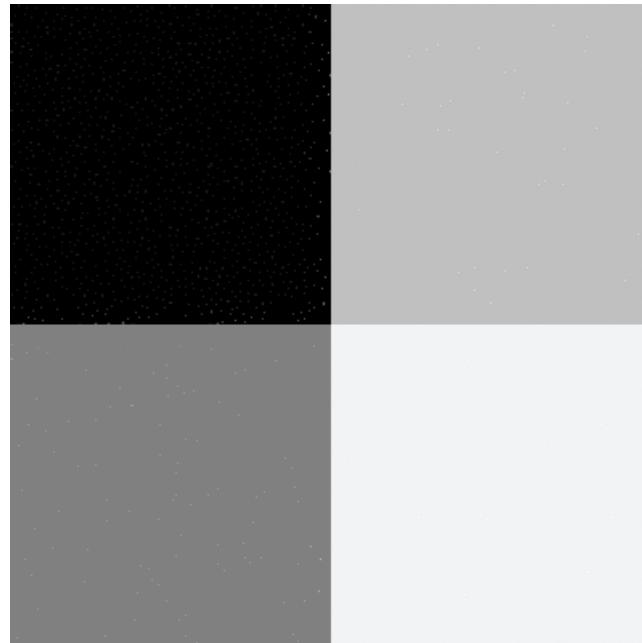
Rysunek 3.5: Użyta dystrybuanta do metody akceptacji-odrzucenia to  $\text{Exp}(\lambda = 20)$ .



Rysunek 3.6: Użyta dystrybuanta do metody akceptacji-odrzucenia to  $\mathcal{LN}(\mu = 1, \sigma = 0,15)$ .

Dla oryginalnego obraz 3.2 został wygenerowany jednorodny szum 3.3. Efekt nałożenia niejednorodnego szumu, wykorzystującego tangensa hiperbolicznego, który był opisany w 1.17 został przedstawiony na obrazie 3.5. Zauważono, że tangens hiperboliczny bardzo szybko zbiega do jedynki, przez co już w ciemnoszarym obszarze jest bardzo mała częstotliwość występowania szumu. Wykorzystując dystrybuantę  $Exp$  z  $\lambda$  równą 20, która była opisana w 1.16, uzyskano niejednorodny szum widniejący na obrazie 3.4. Zauważono szybkie zbieganie prawdopodobieństwa do 1 i słabą intensywność szumu na obszarach o jasności innej niż 0. Do uzyskania niejednorodnego szumu na obrazie 3.6 została użyta dystrybuanta  $\mathcal{LN}$  o parametrach  $\mu = 1$  i  $\sigma = 0,15$ , opisana w 1.15. Prawdopodobieństwo znacznie wolniej dąży do 1, co widać po ciemnoszarym obszarze, gdzie zaobserwowano większą ilość szumu w porównaniu do obrazu 3.4 i 3.5. W rzeczywistości częstotliwość zaszumienia zależy od preferencji artysty. Za pomocą wybranej przez siebie dystrybuanty lub funkcji oraz dobrania odpowiednich parametrów można wpływać na zaszumienie obrazu.

Poniżej zaprezentowano zdjęcia przed i po nałożeniu niejednorodnego szumu.



Rysunek 3.7: Użyta dystrybuanta to  $Exp(\lambda = 1)$ . Modyfikacja jednego piksela. Jasność wyznaczona przy pomocy sąsiadów.

Na obrazie 3.7 widać, jak zmienia się częstotliwość pojawiania się szumów względem jasności obszarów. Przy granicach kwadratów o różnych jasnościach szarości można zauważyć, jak sąsiedzi wpływają na jasność piksela podczas nakładanii szumu.



Rysunek 3.8: Oryginalny obraz.

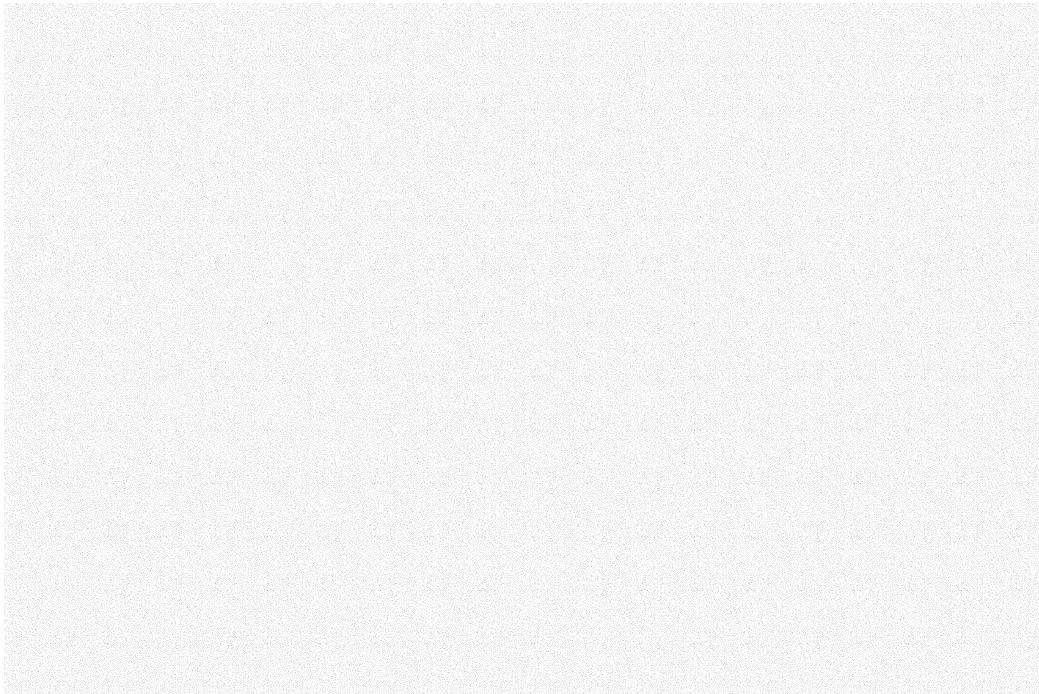


Rysunek 3.9: Użyto dystrybuanty  $Exp(\lambda = 1)$  do modyfikacji jednego piksela. Jasność wyznaczona przy pomocy sąsiadów.

Obrazy mają wymiar  $452 \times 495$ . Na obrazie 3.9 zauważać można, że w okolicach pioruna bardzo rzadko pojawia się szum, natomiast w koronie drzew widać duże zagęszczenie szumów oraz zmianę jasności pikseli na granicach obszarów z różnymi jasnościami.



Rysunek 3.10: Oryginalny pełen obraz.



Rysunek 3.11: Wygenerowany jednorodny szum dla obrazu 3.10.



Rysunek 3.12: Przetworzenie szumu na jednorodny szum z wykorzystaniem  $\text{Exp}(\lambda = 20)$  dla obrazu 3.10.



Rysunek 3.13: Generacja szumu na pełen obraz.

Obrazy 3.10–3.13 mają rozmiary  $3888 \times 2592$ . Jasność została zmodyfikowana również dla sąsiadów piksela, na którego nakładano niejednorodny szum. Obraz 3.12 przedstawia niejednorodny szum. Widać jak gęstość kropek zmniejsza się wraz ze zwiększeniem się jasności obszaru. Zaobserwować można obrrys błyskawicy, który jest ukazany w postaci obszarów z ledwo widocznymi szumami. W obrazie 3.13 w ciemnych obszarach występują ciemnokolorowe plamy. Nie jest to wina programu, lecz artefaktów w oryginalnym zdjęciu 3.10, a nałożony szum tylko je mocniej uwidacznia. Ukazuje to zmianę częstotliwości występowania szumu względem jasności obszaru.



# Podsumowanie

W pracy został zaprezentowany autorski algorytm, który generował płaszczyznę wypełnioną szumem przy użyciu parkietużu. Z wygenerowanego obszaru pobrano 4 segmenty i po przypisaniu do nich kolorów stworzono zestaw kafelków. Z zestawu pobrano  $n$  ilość kafelków i przy pomocy jądra oraz minimalnego błędu cięcia granicznego stworzono kafelki. Ze stworzonych kafelków, używając parkietużu, ułożono płaszczyznę wypełnioną jednorodnymi szumami o tych samych wymiarach co oryginalny obraz. Na koniec za pomocą metody akceptacji odrzucenia i wybranej dystrybuanty akceptowano, lub odrzucano względem jasności nakładanie szumu na piksel. Autorska implementacja dała efekt w postaci obrazów 3.7, 2.15, 3.9. Poprzez zastosowanie parkietużu, proces generowania jednorodnego szum na cały obraz był szybszy niż gdyby generowano całą płaszczyznę, używając tylko próbkowania dysków Poissona, co pokazano na wykresie 3.1. Częstotliwość szumu jest odwrotnie proporcjonalna do jasności pikseli na danym obszarze, czyli uzyskano docelowy efekt.

Nie udało się dobrać odpowiedniej jasności oraz dostosować obszar pokrycia szumu, by szum wyglądał naturalnie. Implementacja nie była optymalna. W przyszłości można zmniejszyć wpływ artefaktów podczas nakładania szumu na obraz, jak miało to miejsce w obrazie 3.13. Można pomyśleć o tworzeniu i wybieraniu zestawów, których kafelki gwarantują nieokresowość. Poprzez ankietyzację i analizę można dobrać parametry dystrybuant wzgółdem pikseli obrazu tak, aby wizualnie obraz był atrakcyjniejszy.



# Bibliografia

- [1] ANTONIADIS, P. How to convert a color from hsl to rgb. <https://www.baeldung.com/cs/convert-color-hsl-rgb>, 2022. Ostatnio odwiedzane 31.12.2023.
- [2] BRIDSON, R. Fast poisson disk sampling in arbitrary dimensions. *SIGGRAPH '07: ACM SIGGRAPH 2007* 07 (2007), 22–es.
- [3] COHEN, M. F., SHADE, J., HILLER, S., DEUSSEN, O. Wang tiles for image and texture generation. *ACM Transactions on Graphics* 22, 3 (2003), 287–294.
- [4] FOPPA, I. M. 1 - D. Bernoulli: A pioneer of epidemiologic modeling (1760). In *A Historical Introduction to Mathematical Modeling of Infectious Diseases*, I. M. Foppa, Ed. Academic Press, Boston, 2017, pp. 1–20.
- [5] GWOSDEK, P., GREWENIG, S., BRUHN, A., WEICKERT, J. Theoretical foundations of gaussian convolution by extended box filtering. *Scale Space and Variational Methods in Computer Vision* 6667 (2012), 447–458.
- [6] KADRY, S. 5 - calculus. In *Mathematical Formulas for Industrial and Mechanical Engineering*, S. Kadry, Ed. Elsevier, Oxford, 2014, pp. 65–111.
- [7] LIMPERT, E., STAHEL, W. A., ABBT, M. Log-normal Distributions across the Sciences: Keys and Clues: On the charms of statistics, and how mechanical models resembling gambling machines offer a link to a handy way to characterize log-normal distributions, which can provide deeper insight into variability and probability—normal or log-normal: That is the question. *BioScience* 51 (2001), 341–352.
- [8] MCCOOL, M., UME, E. F. Hierarchical poisson disk sampling distributions. *Proceedings of the conference on Graphics interface '92* (1992), 94–105.
- [9] NEWSON, A., FARAJ, N., GALERNE, B., DELON, J. Realistic film grain rendering. *Image Processing On Line* 7 (2017), 165–183.
- [10] OSEMAN, N. How film works. <https://neiloseman.com/how-film-works/>, 2020. Ostatnio odwiedzane 21.12.2023.
- [11] SARAVANAN, G., YAMUNA, G., NANDHINI, S. Real time implementation of rgb to hsv/hsi/hsl and its reverse color space models. In *2016 International Conference on Communication and Signal Processing (ICCSP)* (2016), pp. 0462–0466.
- [12] SNEHA, H. 2d convolution in image processing. <https://www.allaboutcircuits.com/technical-articles/two-dimensional-convolution-in-image-processing/>, 2018. Ostatnio odwiedzane 04.01.2024.

- [13] WELLS, W. M. Efficient synthesis of gaussian filters by cascaded uniform filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8*, 2 (1986), 234–239.