Politechnika Wrocławska

Inżynieria Obrazów

Laboratorium nr 6

Dithering i rasteryzacja

Szymon Datko & Mateusz Gniewkowski

szymon.datko@pwr.edu.pl , mateusz.gniewkowski@pwr.edu.pl

Wydział Elektroniki,

Politechnika Wrocławska

semestr letni 2020/2021

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Cel ćwiczenia

1. Zaznajomienie się z problematyką redukcji kolorów w obrazach.
2. Implementacja algorytmu ditheringu w obrazach.
3. Usystematyzowanie informacji na temat potoku graficznego.
4. Zapoznanie się z technikami rasteryzacji obrazu.

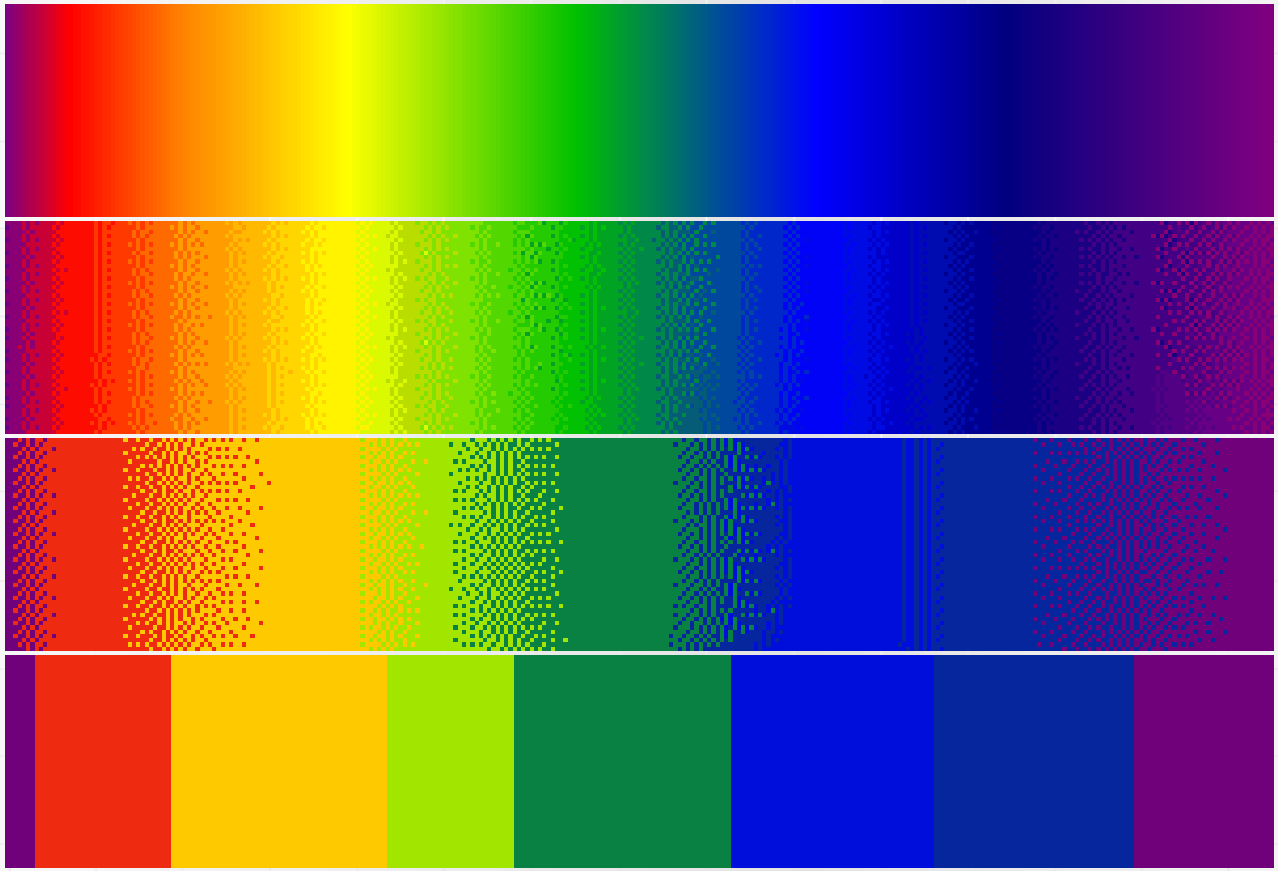
Redukcja palety barw

- Typowo każdą składową koloru reprezentujemy jako jedną z 256 wartości.
  - Jest to tak zwany *true color* – głębia 24-bitowa, 8 bitów na kanał.
  - Daje to w sumie 16 777 216 unikalnych odcieni możliwych do uzyskania.
- Niektóre urządzenia i formaty graficzne są ograniczone w tej kwestii.
- Efektywnie mamy wtedy dostępnych mniej wartości pośrednich odcieni.

```
k = 256
        0  1  2  ...  63  64  65  ...  127  128  129  ...  191  192  193  ...  253  254  255

k = 9
        0        32       64       96       128      160      192      224             255

k = 5
        0                 64                128                192                       255

k = 3
        0                                   128                                        255

k = 2
        0                                                                              255
```

Dithering

- Zastosowanie szumu w celu zniwelowania błędu kwantyzacji.
- Zapobiega efektowi pasmowania przy zredukowanej palecie kolorów.



Źródło obrazów: https://pl.wikipedia.org/wiki/Dithering_(grafika_komputerowa)

Algorytm Floyda–Steinberga

- Kontrolowane rozpraszanie pikseli o kolorach z ograniczonej palety barw.
- Dla każdego przetwarzanego piksela obliczany jest błąd kwantyzacji.
- Wartość błędu jest dodawana do sąsiednich, nieprzetworzonych pikseli.
  - Jeśli bieżący piksel był rozjaśniony, to sąsiednie będą przyciemnione, itd.
- Rozproszenie błędu realizowane jest według macierzy współczynników,

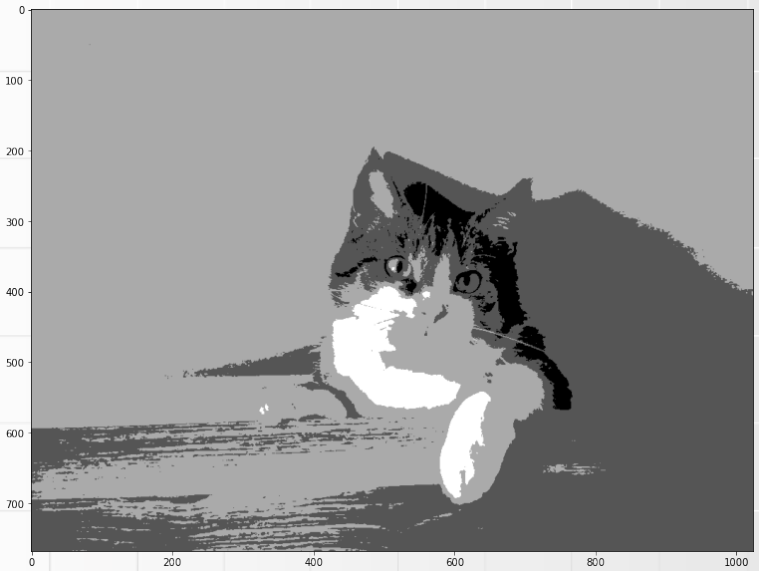
$$\begin{bmatrix} — & * & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} & \frac{1}{16} \end{bmatrix}.$$

```
for each y from top to bottom do
    for each x from left to right do
        oldpixel := pixel[y][x]
        newpixel := find_closest_palette_color(oldpixel)
        pixel[y][x] := newpixel
        quant_error := oldpixel - newpixel
        pixel[y    ][x + 1] := pixel[y    ][x + 1] + quant_error * 7 / 16
        pixel[y + 1][x - 1] := pixel[y + 1][x - 1] + quant_error * 3 / 16
        pixel[y + 1][x    ] := pixel[y + 1][x    ] + quant_error * 5 / 16
        pixel[y + 1][x + 1] := pixel[y + 1][x + 1] + quant_error * 1 / 16
```

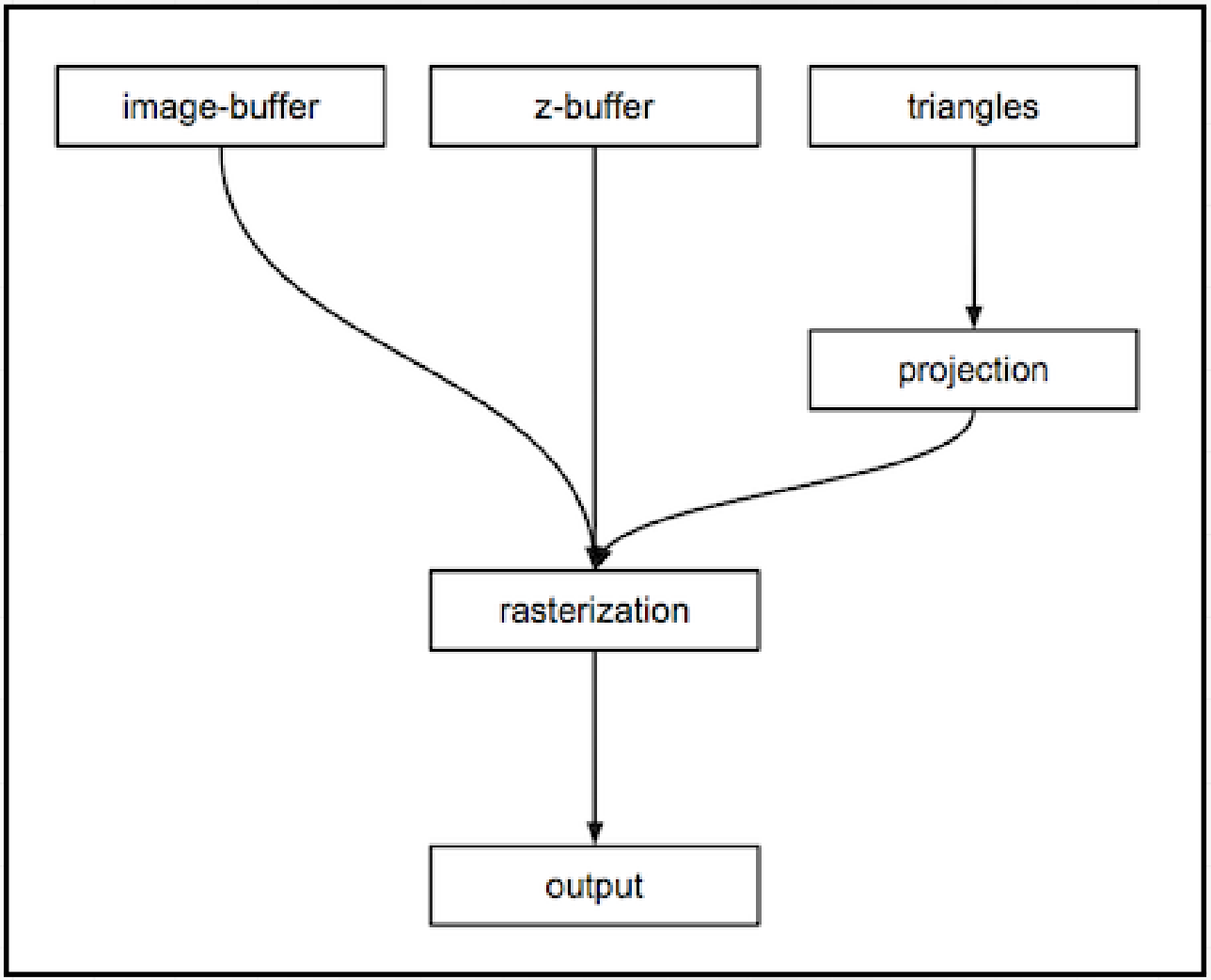
Źródło: https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering

Algorytm Floyda–Steinberga – przykład

- Oba poniższe obrazki składają się jedynie z czterech tych samych kolorów.
- Różni je jedynie ich proporcja ilościowa i rozmieszczenie na obrazku.
- Algorytm można zrealizować także na obrazkach kolorowych.
  - Obliczenia powtarzamy wtedy dla każdej składowej koloru niezależnie.
- **Ważne**: wejście algorytmu stanowi obrazek **przed** redukcją kolorów.



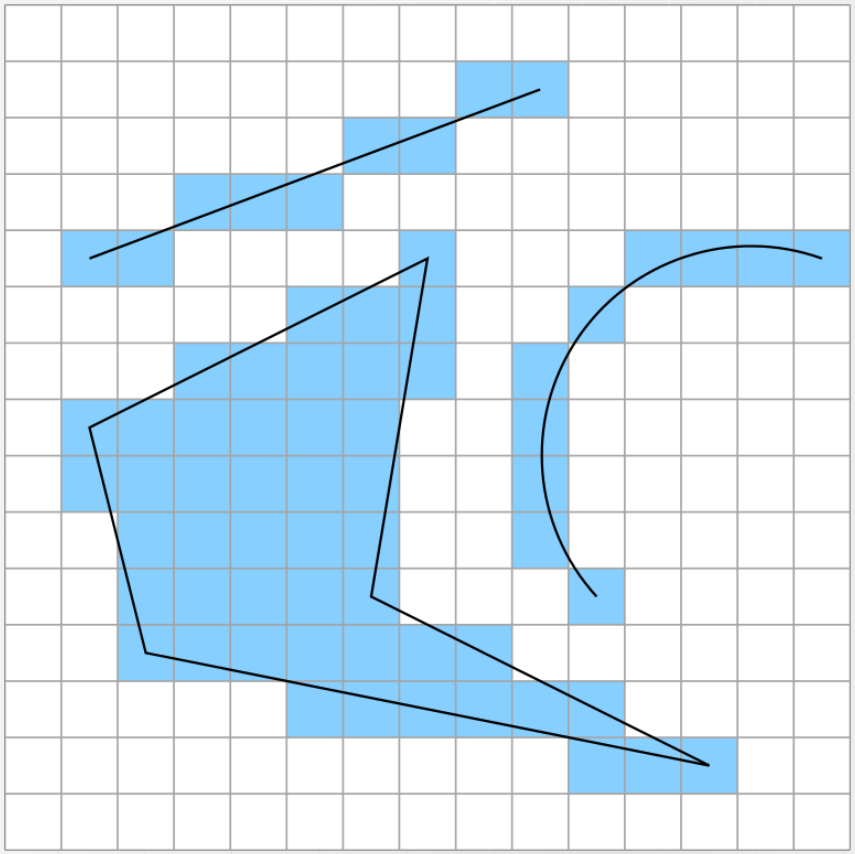
Elementy potoku graficznego



Źródło obrazka:
https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

Rasteryzacja

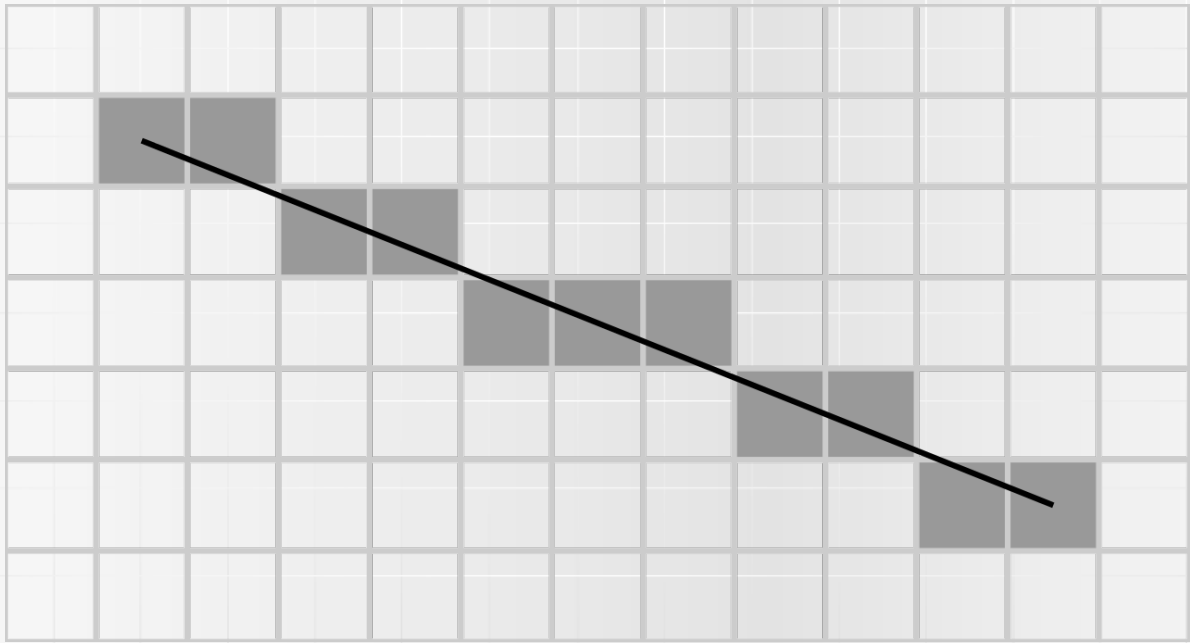
- Proces odwzorowywania kształtu w możliwy do wyświetlenia zbiór pikseli.
- Obecnie realizowany jest głównie sprzętowo przez procesory graficzne.
- Implementacje programowe dotyczą np. edytorów grafiki wektorowej.



Źródło obrazu: https://pl.wikipedia.org/wiki/Rasteryzacja

Algorytm Bresenhama

- Bardzo wydajny algorytm rysowania ciągłych odcinków.
- Opiera się na analizie współczynnika kierunkowego i minimalizacji błędu.
- Wyznaczamy kierunek zmian w osi X i Y oraz dominującą oś dla zmian.
  - W kierunku dominującym zmiana następuje w każdym kroku.
  - Krok w drugim kierunku zależny jest od wyznaczonego błędu.
- Linię rysujemy od punktu początkowego aż do osiągnięcia końca.



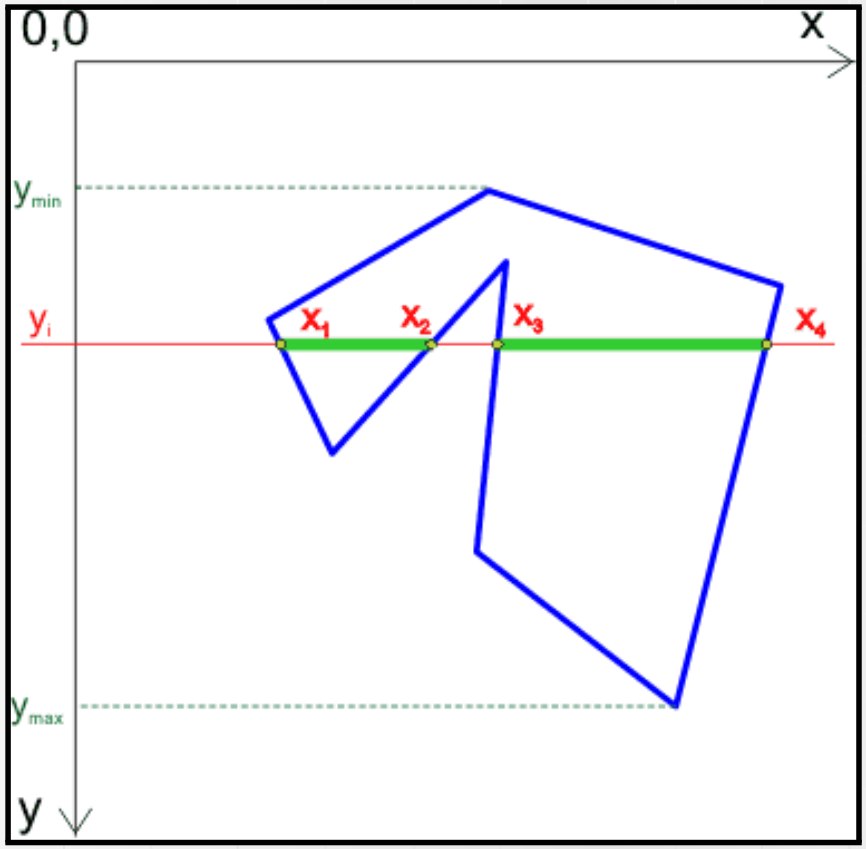
Źródło obrazu: https://en.wikipedia.org/wiki/Bresenham's_line_algorithm

Algorytm Bresenhama – przykładowa implementacja

- Wyznaczenie wielkości pomocniczych,
  - $\Delta X = |X_2 - X_1|$,
  - $\Delta Y = |Y_2 - Y_1|$,
  - $X_i = sign(X_2 - X_1) = \frac{X_2 - X_1}{|X_2 - X_1|}$ lub 0,
  - $Y_i = sign(Y_2 - Y_1) = \frac{Y_2 - Y_1}{|Y_2 - Y_1|}$ lub 0.
- Określenie początkowej wartości błędu,
  - $d = 2 \cdot \Delta Y - \Delta X$, gdy $\Delta X > \Delta Y$,
  - $d = 2 \cdot \Delta X - \Delta Y$, gdy $\Delta Y > \Delta X$.
- Rysowanie w punkcie początkowym $X = X_1$ i $Y = Y_1$.
- Powtarzanie w pętli aż do osiągnięcia punktu docelowego.
  - Krok w kierunku dominującym.
    - $X \mathrel{+}= X_i$ oraz $d \mathrel{+}= 2 \cdot \Delta Y$, gdy $\Delta X > \Delta Y$.
    - $Y \mathrel{+}= Y_i$ oraz $d \mathrel{+}= 2 \cdot \Delta X$, gdy $\Delta Y > \Delta X$.
  - Krok w drugim kierunku, o ile trzeba.
    - $Y \mathrel{+}= Y_i$ oraz $d \mathrel{-}= 2 \cdot \Delta X$, gdy $\Delta X > \Delta Y$ i $d \geq 0$.
    - $X \mathrel{+}= X_i$ oraz $d \mathrel{-}= 2 \cdot \Delta Y$, gdy $\Delta Y > \Delta X$ i $d \geq 0$.

Algorytm przeglądania linii

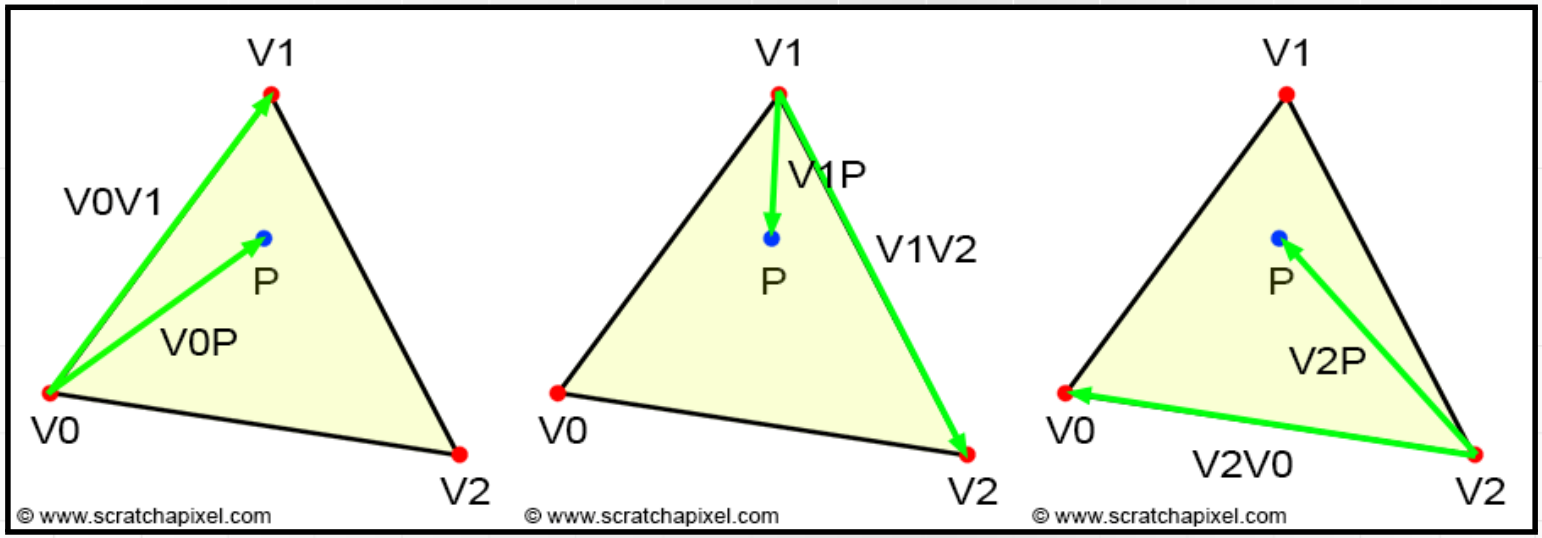
- Jeden z klasycznych pomysłów na rysowanie wypełnionych wielokątów.
- Algorytm staje się kłopotliwy przy dużej liczbie prymitywów.
- Obecnie implementuje się techniki związane z rysowaniem trójkątów.



Źródło obrazu: https://eduinf.waw.pl/inf/utils/002_roz/2008_22.php

Rysowanie trójkątów

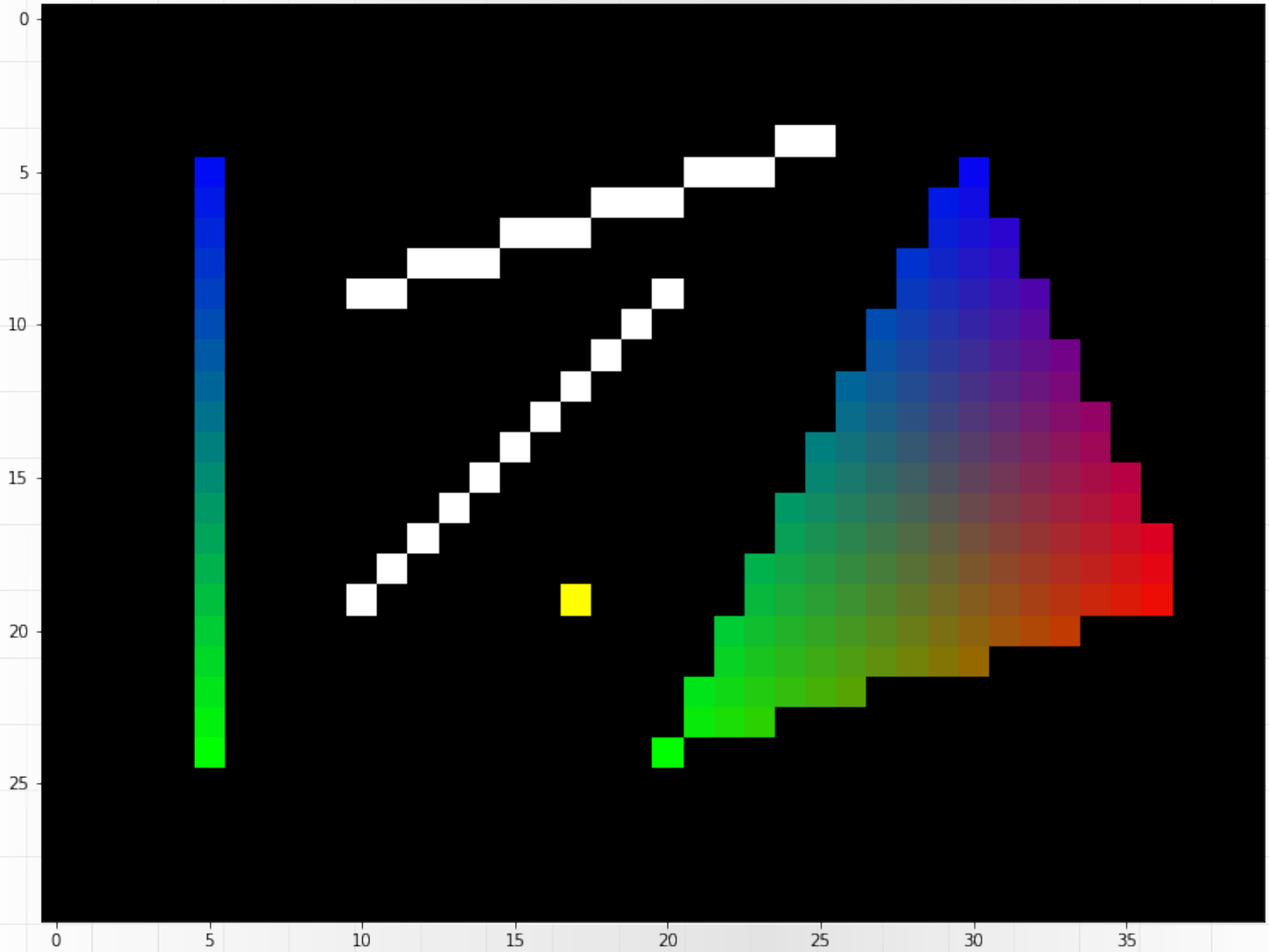
- Trójkąt jest podstawową jednostką renderingu we współczesnym potoku.
- Wykorzystuje się prostotę w określeniu czy dany punkt należy do trójkąta.
- Przykładowy pomysł:
  - Wyznaczyć pola trzech trójkątów, bazując na iloczynie wektorowym w 2D.
  - $Area(A, B, C) = (C.x - A.x) \cdot (B.y - A.y) - (C.y - A.y) \cdot (B.x - A.x)$.
  - Na rysunku: $Area(V0, V1, P)$, $Area(V1, V2, P)$, $Area(V2, V0, P)$.
  - Jeśli znaki (+/-) wyników są takie same to punkt P należy do trójkąta.



Źródło obrazu: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage

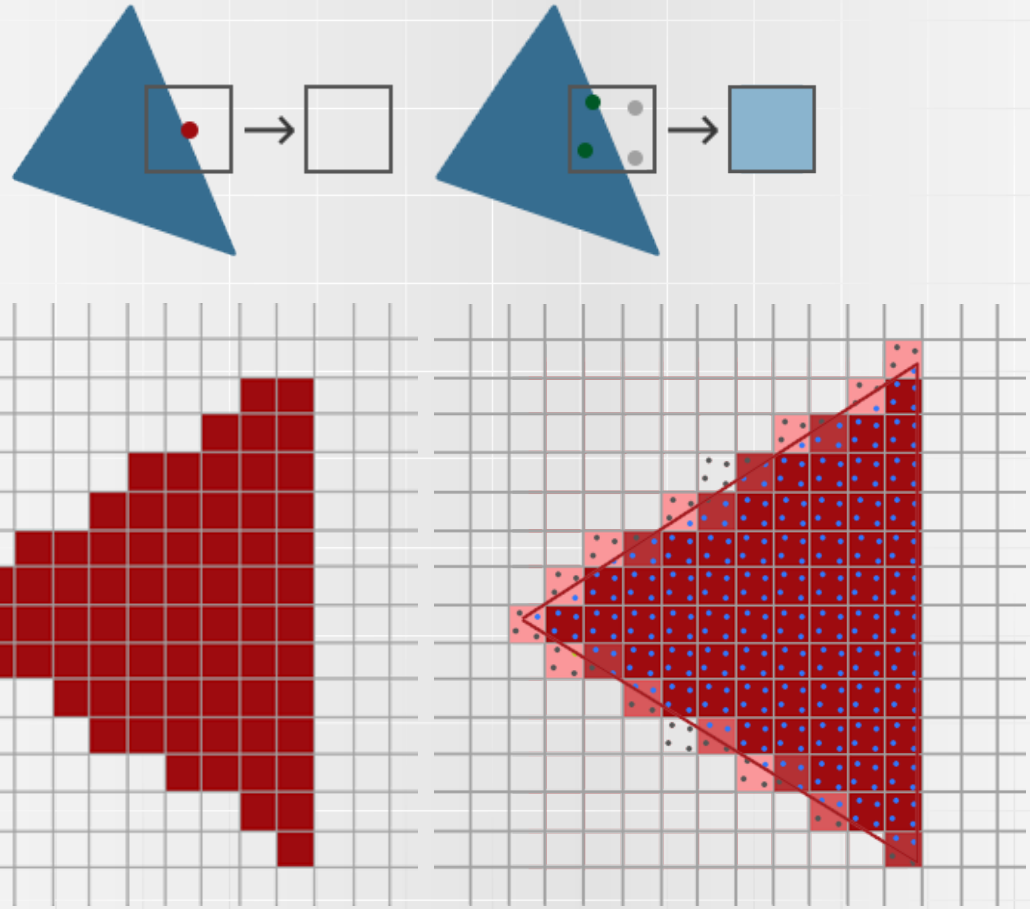
Politechnika Wrocławska

Przykładowy rezultat



Wygładzanie krawędzi

- Dodatkowe zapobieganie ostrym przejściom na krawędziach obiektów.
- Może być zaimplementowane jako integralna część algorytmu rysowania.
  - https://en.wikipedia.org/wiki/Xiaolin_Wu's_line_algorithm



Źródło obrazów: https://learnopengl.com/Advanced-OpenGL/Anti-Aliasing

Politechnika Wrocławska

Koniec wprowadzenia.

Zadania do wykonania...

Zadania do wykonania (1)

Na ocenę **3.0** należy zaimplementować dithering w skali szarości.

Wskazówki:

- odwzorować w języku Python kod ze slajdu 5,
- wygodnie będzie pracować na wartościach typu `uint8`,
- funkcja `find_closest_palette_color()` realizuje redukcję palety barw,
  - w najprostszym układzie wystarczy zaokrąglić wartości w macierzy pikseli,
  - `return round(value / 255) * 255`,
- dodanie błędu kwantyzacji może spowodować wyjście poza zakres typu,
  - należy odpowiednio się przed tym zabezpieczyć i przyciąć wartości,
  - w przeciwnym wypadku zaobserwujemy pojedyncze *bad pixele*,
- należy też uważać na zakres indeksów przy odwołaniach do tablicy pikseli.

Zadania do wykonania (2)

Na ocenę **3.5** należy uwzględnić kolory w algorytmie Floyda–Steinberga.

Wskazówki:

- uwagi i kroki do wykonania są analogiczne jak w zadaniu poprzednim,
- dodatkowo umożliwić wybór ile wartości składowych chcemy zachować,
  - domyślnie mają to być dwie wartości dla każdej składowej – 0 i 255,
  - `return round((k - 1) * value / 255) * 255 / (k - 1)`,
- należy obliczyć błąd kwantyzacji osobno dla każdej składowej koloru,
- dla porównania wyświetlić jak wygląda obrazek po samej redukcji barw,
- skuteczność algorytmu potwierdzić prezentując histogram kolorów.

Zadania do wykonania (3)

Na ocenę **4.0** należy zaimplementować rysowanie jednokolorowej linii i trójkąta.

Wskazówki:

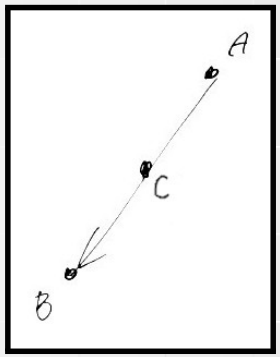
- wykorzystać dostarczoną funkcję `draw_point()` do rysowania,
- rysowanie linii zrealizować np. algorytmem Bresenhama,
- rysowanie trójkąta polega na testowaniu które punkty leżą w jego środku,
  - przykładowy pomysł na sprawdzanie przynależności opisano na slajdzie 12,
- dla trójkąta warto wyznaczyć obszar ograniczający wokół niego,
  - skutecznie ogranicza to liczbę punktów, które musimy przetestować,
  - tak zwany *bounding box*, na przykład dla trójkąta ABC:
  - $x_{min} = min(A.x, B.x, C.x)$, $x_{max} = max(A.x, B.x, C.x)$,
- dopuszczalna jest realizacja innych algorytmów, dających sensowny wynik.

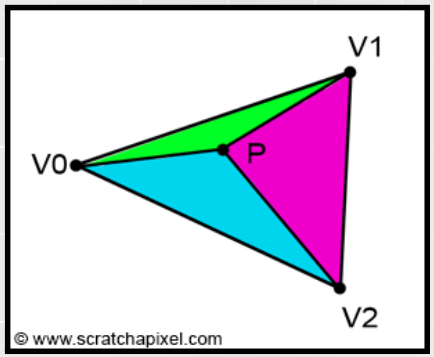
Zadania do wykonania (4)

Na ocenę **4.5** należy wprowadzić interpolację koloru w rysowanej linii i trójkącie.

Wskazówki:

- zakładamy, że kolor jest podany z każdym wierzchołkiem (jak położenie),
- w przypadku linii można wykorzystać interpolację liniową koloru,
  - $\vec{C} = \vec{A} + t \cdot (\vec{B} - \vec{A})$, dla $t \in [0; 1]$,
  - t określa wtedy postęp w rysowaniu linii ($0 =$ początek, $1 =$ koniec),
- w przypadku trójkąta wykorzystujemy ważenie względem pól trójkątów,
  - $\vec{C}_P = \frac{\lambda_0}{\lambda} \cdot \vec{V0} + \frac{\lambda_1}{\lambda} \cdot \vec{V1} + \frac{\lambda_2}{\lambda} \cdot \vec{V2}$,
  - $\lambda = Area(V0, V1, V2)$,
  - $\lambda_0 = Area(V0, V1, P)$, $\lambda_1 = Area(V1, V2, P)$, $\lambda_2 = Area(V2, V0, P)$.





Źródło obrazu: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage

Politechnika Wrocławska

Zadania do wykonania (5)

Na ocenę **5.0** należy dodać wygładzanie krawędzi w generowanym obrazie.

Wskazówki:

- doskonale sprawdzi się **SSAA** – ang. *Super Sampling Anti-Aliasing*,
- należy wygenerować obraz roboczy w wyższej rozdzielczości,
  - wystarczy wysokość, szerokość i wszystkie współrzędne przemnożyć razy 2,
  - na tym etapie nie powinno być potrzeby modyfikacji funkcji rysujących,
  - zmianna powinna wiązać się jedynie z wywołaniami funkcji rysujących,
  - warto w tym miejscu wprowadzić dodatkową zmienną, np. `scale = 2`,
- obraz wynikowy uzyskać przez przeskalowanie w dół obrazu roboczego,
  - 1 piksel obrazu wynikowego to 4 uśrednione piksele obrazu roboczego.