# Interpolacja 2D

Krzysztof Zając

07.01.2025

## Zadanie

Zadanie polega na modyfikacji kodu dostarczonego poniżej w taki sposób, aby wykonać interpolację obrazu w skali szarości 2-wymiarową funkcją jądrową. Funkcja `kernel` powinna zawierać implementację funkcji wektorowej: $k : \mathbb{R}^2 \to \mathbb{R}^1$. Funkcja `image_interpolate` powinna zwracać obraz o zadanym rozmiarze, większym niż obraz wejściowy `image`. Linijki oznaczone `#C` oraz ciało funkcji `kernel` powinny zostać zmienione, żeby poprawnie wykonać zadanie.

### 2-wymiarowe funkcje jądrowe

$$H_s(x, c, W) = \begin{cases} 1 \text{ gdy } 0 < (x_1 - c_1) < W \text{ oraz } 0 < (x_2 - c_2) < W \\ 0 \text{ w przeciwnym razie} \end{cases}$$

$$H_n(x, c, W) = \begin{cases} 1 \text{ gdy } -W/2 < (x_1 - c_1) < W/2 \text{ oraz } -W/2 < (x_2 - c_2) < W/2 \\ 0 \text{ w przeciwnym razie} \end{cases}$$

### Przykładowa implementacja

```
import numpy as np
from numpy.typing import NDArray


def kernel(points: NDArray, offset: float | NDArray, width: float):  # C
    ...


def image_interpolate2d(image: NDArray, ratio: int) -> NDArray:
    """
    Interpolate image using 2D kernel interpolation

    :param image: grayscale image to interpolate as 2D NDArray
    :param ratio: up-scaling factor

    :return: interpolated image as 2D NDArray
    """
    w = 1  # C
    target_shape = None  # C
    image_grid = None  # C
    interpolate_grid = None  # C

    kernels = []
    for point, value in zip(image_grid, image.ravel()):
        kernel = value * kernel(interpolate_grid, offset=point, width=w)
        kernels.append(kernel.reshape())

    return np.sum(np.asarray(kernels), axis=0)
```

# Zasady Zaliczenia

Zadanie należy wykonać w trakcie zajęć i oddać je jako sam kod, wraz z potrzebnymi komentarzami. Plik (lub kilka plików) wysłać jako archiwum (ZIP) lub poprzez repozytorium `github`. Maksymalny czas na oddanie to 08.01.2025 do północy, co daje dodatkowy dzień na dokończenie pracy. Każdy dodatkowy dzień opóźnienia skutkuje obniżeniem oceny o 0.5, ale nie poniżej 3.0. W przypadku nieobecności na zajęciach, zasady zaliczenia należy **ustalić indywidualnie**.

## Kryteria Oceniania

- 3.0 - dokończyć implementacje podanej funkcji do interpolacji 2D korzystając z jądra "sample hold" ($H_s$) lub "nearest neighbor" ($H_n$).
- 4.0 - zaimplementować dodatkowe wybrane funkcje jądrowe do interpolacji 2D, odcinany sinus, jądro liniowe lub funkcję Keysa.
- 5.0 - rozszerzyć implementację na kolorowe obrazy.

# Źródła

- https://en.wikipedia.org/wiki/Bicubic-interpolation
- http://verona.fi-p.unam.mx/boris/practicas/CubConvInterp.pdf