Kompresja H265 HEVC

Autor: Karol Pisarski

W projekcie zrealizowano kompresję wideo przy użyciu bibliotek zewnętrznych w języku Python, w szczególności korzystając z biblioteki **imageio\_ffmpeg**, która integruje narzędzie **FFmpeg**. FFmpeg to popularne narzędzie open-source służące do przetwarzania multimediów, w tym kodowania i dekodowania wideo.

# Dostarczone materiały

Podczas realizacji projektu stwierdzono, że napisana wcześniej biblioteka przez autora poprzedniej pracy nie jest zbytnio użyteczna i czytelna. To zwiększa ryzyko błędnie przeprowadzonych badań, dlatego podjęto decyzję o zaimplementowaniu kodu w języku Python, który zniweluje to ryzyko i umożliwi przeprowadzenie szerszych badań.

Wraz z projektem od prowadzącego otrzymano materiały. Niestety, po głębszej analizie okazało się, że większość z dostarczonych zdjęć do testowania była już skompresowana w formacie .png. Kompresja zdjęć mogła wpłynąć na jakość wyników, co stanowiło istotny problem.

Kolejnym problemem były obrazy o nazwach 5.png, 6.png, 7.png, 8.png, 13.png, które miały niepoprawne rozmiary, co uniemożliwiało ich zakodowanie i dalsze użycie w badaniach.

Podczas badań wybrano trzy główne ścieżki tworzenia siatek z obrazu:

1. Rzędowa – polegająca na analizie i przetwarzaniu obrazu rząd po rzędzie.
2. Spiralna – oparta na przetwarzaniu obrazu w układzie spiralnym.
3. Krzywa Hilberta – używająca złożonej krzywej przestrzennej do kodowania obrazu.

Do badania dołączono również nowe cztery zdjęcia, które pozwoliły na poprawę zdolności oceny wyników eksperymentu, dzięki czemu uzyskano bardziej wiarygodne dane do analiz.

# Biblioteka zewnętrzna:

Podczas projektu skorzystano z bibliotek zewnętrznych w języku Python opisanych poniżej:

## Opis biblioteki

Biblioteka imageio\_ffmpeg jest używana do integracji z FFmpeg, narzędziem open-source do przetwarzania multimediów, które obsługuje szeroki zakres operacji na plikach wideo i audio, w tym kodowanie i dekodowanie wideo. FFmpeg jest powszechnie stosowany w aplikacjach multimedialnych ze względu na swoją wszechstronność i wydajność.

## Instalacja

Aby zainstalować imageio\_ffmpeg, można użyć następującego polecenia:

sh

pip install imageio[ffmpeg]

## Zastosowanie w projekcie

W projekcie wykorzystano bibliotekę imageio\_ffmpeg do kodowania i dekodowania obrazów jako wideo za pomocą kodeka H.265 (HEVC). Poniżej przedstawiono szczegóły implementacji.

## Kodowanie wideo

Do kodowania obrazów jako wideo użyto następującej metody:

python

import imageio\_ffmpeg as ffmpeg

def encode\_as\_video(self, tiles, output\_path):

input\_path = 'temp\_input\_frames'

if not os.path.exists(input\_path):

os.makedirs(input\_path)

# Save tiles as individual frames

for i, tile in enumerate(tiles):

tile\_resized = cv2.resize(tile, (self.frame\_size, self.frame\_size), interpolation=cv2.INTER\_NEAREST)

cv2.imwrite(f'{input\_path}/frame\_{i:04d}.png', cv2.cvtColor(tile\_resized, cv2.COLOR\_RGB2BGR))

# Use imageio-ffmpeg to encode frames to video with extra options

cmd = [

ffmpeg.get\_ffmpeg\_exe(), '-y', '-framerate', '1', '-i',

f'{input\_path}/frame\_%04d.png', '-c:v', self.codec

] + self.extra\_options + [output\_path]

os.system(' '.join(cmd))

# Clean up temporary frames

for file in os.listdir(input\_path):

os.remove(os.path.join(input\_path, file))

os.rmdir(input\_path)

W powyższym kodzie:

* Kafelki obrazu są zapisywane jako indywidualne klatki PNG.
* Następnie klatki te są kodowane do pliku wideo z użyciem FFmpeg i kodeka H.265.

## Dekodowanie wideo

Do dekodowania wideo z powrotem na obrazy użyto następującej metody:

python

def decode\_video\_to\_image(self, video\_path, grid\_size):

import tempfile

temp\_dir = tempfile.mkdtemp()

cmd = [

ffmpeg.get\_ffmpeg\_exe(), '-i', video\_path,

f'{temp\_dir}/frame\_%04d.png'

]

os.system(' '.join(cmd))

frames = []

for file in sorted(os.listdir(temp\_dir)):

if file.endswith('.png'):

frame = cv2.imread(os.path.join(temp\_dir, file))

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frames.append(frame\_rgb)

for file in os.listdir(temp\_dir):

os.remove(os.path.join(temp\_dir, file))

os.rmdir(temp\_dir)

# Ensure frames list has enough frames to fill the grid

total\_frames = grid\_size \* grid\_size

while len(frames) < total\_frames:

frames.append(np.zeros\_like(frames[0]))

# Reconstruct image from frames

height, width, \_ = frames[0].shape

reconstructed\_image = np.zeros((grid\_size \* height, grid\_size \* width, 3), dtype=np.uint8)

i = 0

for y in range(grid\_size):

for x in range(grid\_size):

reconstructed\_image[y\*height:(y+1)\*height, x\*width:(x+1)\*width, :] = frames[i]

i += 1

return reconstructed\_image

W powyższym kodzie:

* Wideo jest dekodowane na zestaw klatek PNG.
* Klatki te są następnie rekonstruowane w pełny obraz.

# **Kompresja bezstratna HEVC**

Dzięki wyżej wymienionemu problemu, już zakodowanych zdjęć w formacie .png, niesety badane współczynniki dla tych zdjęć nie były wysokie i ciężko jest stwierdzić ich efektywność.

Podstawowe kompresje

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa obrazu | Współczynnik kompresji JPEG | Współczynnik kompresji PNG | Współczynnik kompresji JPEG 2000 |
| 1.png | 2,74 | 1,01 | 0,98 |
| 2.png | 2,74 | 1,01 | 1,05 |
| 3.png | 2,8 | 1,02 | 0,86 |
| 4.png | 3,42 | 1,03 | 0,77 |
| 9.png | 2,84 | 1,02 | 0,75 |
| 10.png | 2,85 | 1,05 | 0,84 |
| 11.png | 2,24 | 1 | 1,02 |
| 12.png | 2,27 | 1,03 | 1,22 |
| 14.png | 2,12 | 1,00 | 1,12 |
| 15.png | 2,14 | 1,02 | 0,95 |
| 16.png | 1,5 | 1,69 | 127,20 |
| 20.bmp | 4,93 | 24,79 | 2,50 |
| Canon-5DMarkII-Shotkit-4.CR2 | 4,66 | 1,96 | 1,51 |
| PIA04230.tif | 4,01 | 1,98 | 1,90 |
| PIA26149.tif | 0,93 | 4,98 | 1,52 |
| sunflower-field.bmp | 4,59 | 2,23 | 2,58 |

Kompresja przy użyciu enkodera H265 HEVC.

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa obrazu | Współczynnik kompresji, dla algorytmu rzędowego | Współczynnik kompresji, dla algorytmy spiralnego | Współczynnik kompresji dla algorytmu hilberta |
| 1.png | 1,79 | 1,79 | 1,78 |
| 2.png | 1,63 | 1,62 | 1,63 |
| 3.png | 1,3 | 1,3 | 1,3 |
| 4.png | 1,69 | 1,69 | 1,68 |
| 9.png | 1,12 | 1,12 | 1,11 |
| 10.png | 1,5 | 1,52 | 1,52 |
| 11.png | 1,44 | 1,41 | 1,43 |
| 12.png | 2,25 | 2,25 | 2,24 |
| 14.png | 1,98 | 1,94 | 1,95 |
| 15.png | 1,49 | 1,48 | 1,47 |
| 16.png | 9,91 | 9,91 | 9,91 |
| 20.bmp | 33,66 | 8,6 | 9,01 |
| Canon-5DMarkII-Shotkit-4.CR2 | 2,99 | 2,91 | 2,94 |
| PIA04230.tif | 3,02 | 2,74 | 2,96 |
| PIA26149.tif | 1,33 | 1,33 | 1,34 |
| sunflower-field.bmp | 4,74 | 4,76 | 4,75 |

Tabela 1. – Wyniki kompresji dla poszczególnych obrazów.

# **Kompresja stratna**

## Współczynnik kompresji 10:1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nazwa obrazu | Współczynnik CRF | Współczynnik kompresji, dla algorytmu rzędowego | Współczynnik kompresji, dla algorytmy spiralnego | Współczynnik kompresji dla algorytmu hilberta |
| 1.png |  |  |  |  |
| 2.png |  |  |  |  |
| 3.png |  |  |  |  |
| 4.png |  |  |  |  |
| 9.png |  |  |  |  |
| 10.png | 18 | 9,66 | 9,72 | 9,66 |
| 11.png |  |  |  |  |
| 12.png |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| Nazwa obrazu | PSNR | SSIM |
| 1.png |  |  |
| 2.png |  |  |
| 3.png |  |  |
| 4.png |  |  |
| 9.png |  |  |
| 10.png | 38,45 | 9,51 |
| 11.png |  |  |
| 12.png |  |  |

## Współczynnik kompresji 30:1

## Współczynnik kompresji 100:1

# Bibliografia:

* <https://shotkit.com/free-raw-photos/>