# Zapoznanie się ze środowiskiem OpenCL

Jan Rosa 06.03.2025

## CLinfo

**Nazwa urządzenia**:

- **Intel CPU**: `cpu-haswell-Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz`
- **NVIDIA GPU**: `NVIDIA GeForce RTX 2070 SUPER`
- **AMD CPU**: `Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz`

- **Typ urządzenia**:

- **Intel CPU**: `CL_DEVICE_TYPE_CPU`
- **NVIDIA GPU**: `CL_DEVICE_TYPE_GPU`
- **AMD CPU**: `CL_DEVICE_TYPE_CPU`

- **Maksymalna częstotliwość zegara**:

- **Intel CPU**: `4900 MHz`
- **NVIDIA GPU**: `1785 MHz`
- **AMD CPU**: `800 MHz`

- **Maksymalna alokacja pamięci**:

- **Intel CPU**: `8589934592 bajtów` (8 GB)
- **NVIDIA GPU**: `2088386560 bajtów` (~2 GB)
- **AMD CPU**: `8392171520 bajtów` (~8 GB)

- **Rozmiar pamięci globalnej**:

- **Intel CPU**: `31421202432 bajtów` (31,4 GB)
- **NVIDIA GPU**: `8353546240 bajtów` (~8 GB)
- **AMD CPU**: `33568686080 bajtów` (~33,6 GB)

## Box Filter

### 1. Przegląd

Filtr pudełkowy (**Box Filter**) wygładza obraz, redukując różnice intensywności między pikselami. Jest stosowany do **usuwania szumów i poprawy jakości obrazu**.

## 2. Implementacja

1. **BoxFilter Separable** – filtracja pozioma i pionowa, z optymalizacją **loop unrolling**, zwiększa wydajność o **10-20%**.
    - **Osiąga 180 FPS** dla obrazu **1024×1024** (filtr 9×9) na **ATI Radeon HD 5770**.
2. **BoxFilter z SAT** – wykorzystuje **Summed-Area Tables (SAT)** do szybkiej filtracji.
    - Po wygenerowaniu SAT filtr wymaga tylko **4 odczytów z pamięci**.
    - **250 FPS dla obrazu 1024×1024**, niezależnie od rozmiaru filtra.

## 3. Wydajność

SAT zapewnia stały czas obliczeń niezależnie od rozmiaru filtra, co daje **znaczącą przewagę nad innymi metodami wygładzania obrazu**.

## 4. Zalecenia dotyczące użycia

Wartość SAT wymaga $\mathbf{\log_2(w) + \log_2(h) + Pi}$ bitów precyzji.

- **Obraz 256×256 (8-bitowe wartości) → 24-bitowa SAT**.
- **Maksymalny obsługiwany rozmiar obrazu**: **4096×4096**.

## 5. Czas działania

### Wersja SAT

| Iterations | Time on CPU (sec) | Time on GPU (sec) |
|---|---|---|
| 10 | 0.0944009 | Not available |
| 200 | 0.094337 | Not available |
| 500 | 0.0947459 | Not available |

### Wersja Separable

| Iterations | Time on CPU (sec) | Time on GPU (sec) |
|---|---|---|
| 10 | 0.0527737 | Not available |
| 200 | 0.0536865 | Not available |
| 500 | 0.0528774 | Not available |

**Niestety, brak jest danych o czasie na GPU, ponieważ system przełączył się na CPU w każdym przypadku z powodu braku wykrycia GPU lub błędów kompilacji kernela.**

---

# HistogramAtomics

## 1. Przegląd

Obliczanie histogramu obrazu poprzez **podział wartości intensywności na 256 przedziałów** z wykorzystaniem **operacji atomowych**.

## 2. Implementacja

1. **Kernel skalarowy** – zoptymalizowany dla architektury **GCN**.
2. **Kernel wektorowy** – lepsza wydajność na **VLIW i starszych kartach**.
    - Wybór wersji: **--scalar** lub **--vector**.
    - **Automatyczny dobór optymalnej szerokości wektora**.

## 3. Wydajność

Zastosowanie operacji atomowych pozwala na **efektywne obliczanie histogramu w środowisku równoległym**, ale może prowadzić do konfliktów dostępu do pamięci.

## 4. Zalecenia dotyczące użycia

Optymalizacja dostępu do pamięci i wybór odpowiedniego typu kernela (**skalarowy lub wektorowy**) poprawia wydajność na różnych architekturach GPU.

## 5. Czas działania

| Iterations | Time on CPU (sec) | Time on GPU (sec) |
|---|---|---|
| 10 | 0.0663259 (Setup) + 0.0462484 (Avg. kernel) = **0.1125743** | 0.0764241 (Setup) + 0.000286499 (Avg. kernel) = **0.0767106** |
| 200 | 0.066354 (Setup) + 0.0471032 (Avg. kernel) = **0.1134572** | 0.190143 (Setup) + 0.000300975 (Avg. kernel) = **0.190443975** |
| 500 | 0.0660149 (Setup) + 0.0462284 (Avg. kernel) = **0.1122433** | 0.0825142 (Setup) + 0.00028427 (Avg. kernel) = **0.08279847** |

---

# ImageOverlap

## 1. Przegląd

Nakładanie dwóch obrazów poprzez **sumowanie wartości RGB** każdego piksela.

## 2. Implementacja

1. Załadowanie obrazu do GPU za pomocą **clCreateImage**.
2. Wypełnienie fragmentów obrazu kolorami przy użyciu **clEnqueueFillImage**.
3. Nakładanie obrazów – sumowanie wartości **RGB** pikseli.
4. Zarządzanie synchronizacją operacji za pomocą **clEnqueueMarkerWithWaitList**.

## 3. Wydajność

Wykorzystanie **clCreateImage** i **clEnqueueFillImage** optymalizuje dostęp do pamięci, a operacje równoległe przyspieszają sumowanie wartości pikseli.

## 4. Zalecenia dotyczące użycia

Wymagany **OpenCL 1.2**, aby poprawnie obsługiwać operacje na obrazach i zapewnić odpowiednią wydajność.

## 5. Czas działania

| Iterations | Time on CPU (sec) | Time on GPU (sec) |
|---|---|---|
| 10 | 0.0364973 | 0.0490276 |
| 200 | 0.0365157 | 0.102426 |
| 500 | 0.036479 | 0.0441826 |

---

# Sobel Filter

## 1. Przegląd

Filtr Sobela służy do **detekcji krawędzi** poprzez obliczenie **gradientu intensywności pikseli**.

## 2. Implementacja

1. **Dwa 3×3 filtry** stosowane oddzielnie dla kierunku poziomego (**Dx**) i pionowego (**Dy**).
2. Obliczenie gradientu według wzoru: S=Dx2+Dy2S = \sqrt{Dx^2 + Dy^2}
3. **Bufor stały** optymalizuje odczyt sąsiednich pikseli.

## 3. Wydajność

Wykorzystanie **stałej pamięci** do przechowywania danych wejściowych minimalizuje liczbę odwołań do pamięci globalnej, co poprawia szybkość działania algorytmu.

## 4. Zalecenia dotyczące użycia

•**Optymalne wejście**: duże obrazy, np. **2400×1600**, dla lepszej jakości detekcji krawędzi.**5.**

## 5. Czas działania

| Iterations | Time on CPU (sec) | Time on GPU (sec) |
|---|---|---|
| 10 | 0.0239002 | 0.044098 |
| 200 | 0.0269372 | 0.0438288 |
| 500 | 0.0242875 | 0.0494904 |