Analiza algorytmów przeszukiwania obrazów

Wstęp

Zagadnienie omawiane w analizie dotyczy wydajności wyszukiwania wyznaczonych elementów na zdjęciu, czyli tak naprawdę wyszukiwaniu bitmapy w bitmapie. W badaniu zostało to zaprezentowane na podstawie 5 algorytmów przeszukujących. Każdy z nich poddany został pomiarowi czasów w przypadkach pesymistycznych i optymistycznych oraz sprawdzono ich wydajność na szeregu bitmap o różnych wymiarach. Środowisko w którym przeprowadzono eksperyment to platforma .NET, IDE Visual Studio 2019 wersja 16.8.3, .NET Framework 3.8.04084, a urządzenia, które dokonywały pomiarów to: (AMD Athlon™ X4 Quad-Core, 8GB RAM, Windows 10) oraz (AMD Ryzen 5 4600H, 8 GB RAM, Windows 10).

Opis algorytmów

- GetPixel

Algorytm najprostszy w budowie, ponieważ zwyczajnie przechodzi przez współrzędne bitmapy bazowej z wyłączeniem obszarów skrajnych. Obszar ten zajmuje sporą część głównego obrazu, gdyż jest zależny od wielkości poszukiwanego elementu. Prościej to ujmując wysokość przeszukiwanego obszaru bazowej bitmapy jest równa jej wysokości odjętej od wysokości bitmapy poszukiwanej plus jeden piksel, dla szerokości wygląda to analogicznie. Dzięki dwóm pętlą algorytm operuje w dwóch wymiarach przesuwając się po kolejnych pikselach, a na każdym z nich rozpoczyna porównywanie z bitmapą szukanego elementu za pośrednictwem wyciągania wartości piksela metodą „GetPixel”.

- GetPixelBorder

Jest to bardzo podobna implementacja do algorytmu GetPixel, ponieważ zawiera tę samą metodę wyciągania wartości pikseli. Znaczącą różnicą jest jednak sposób sprawdzania zgodności szukanej bitmapy z wyznaczonym obszarem na bazowej bitmapie. Nie dzieje się to poprzez kolejne sprawdzanie bitów od lewej do prawej, jednak zaczyna się to od sprawdzenia ramki bitmapy szukanej. Jeżeli jest ona zgodna z ramką bazową to algorytm przechodzi do sprawdzania środkowej części z wyłączeniem już przeszukanych ramek.

- MemoryArray

Algorytm ten zaczyna swoje działanie od stworzenia dwóch obiektów BitmapData na podstawie bitmapy bazowej oraz poszukiwanej. Dzięki temu możliwe jest zablokowanie danych obrazu w pamięci procesora, co dzieje się za pomocą metody „LockBits”. Po umieszczeniu danych w pamięci zapisywane są one do tablicy dwuwymiarowej jaggedarray, czyli tablicy w której wymiary są inicjalizowane osobno (np. int[][]), co zwiększa szybkość pracy. Dalsza część działania algorytmu jest podobna do poprzednich, przeszukiwane są kolejne bity zapisane w tablicy z wyłączeniem obszaru skrajnego, następnie porównywane z bitami elementu szukanego zapisanego w osobnej tablicy. Gdy liczba z tablicy odpowiadająca bitowi obrazu nie zgadza się z liczbą z tablicy elementu szukanego pętle są przerywane i kontynuowane jest sprawdzanie następnych współrzędnych.

- InsideMemory

  W tej wariancji algorytmu wykorzystywane są również obiekty BitmapData, jednak że dane z pamięci procesora nie są zapisywane od razu do dwuwymiarowych tablic. Przy każdej iteracji pierwszej pętli poruszającej się względem osi Y nadpisywana jest nowa tablica z linią wartości liczbowych bitów obrazu. Po trafieniu na identyczną linię wartości bitów elementu szukanego tworzone oraz nadpisywane są następne linie wartości bitów do sprawdzenia. Ostatecznie dopiero przed zwróceniu wyniku pamięć systemowa jest zwolniona z obu obrazów.

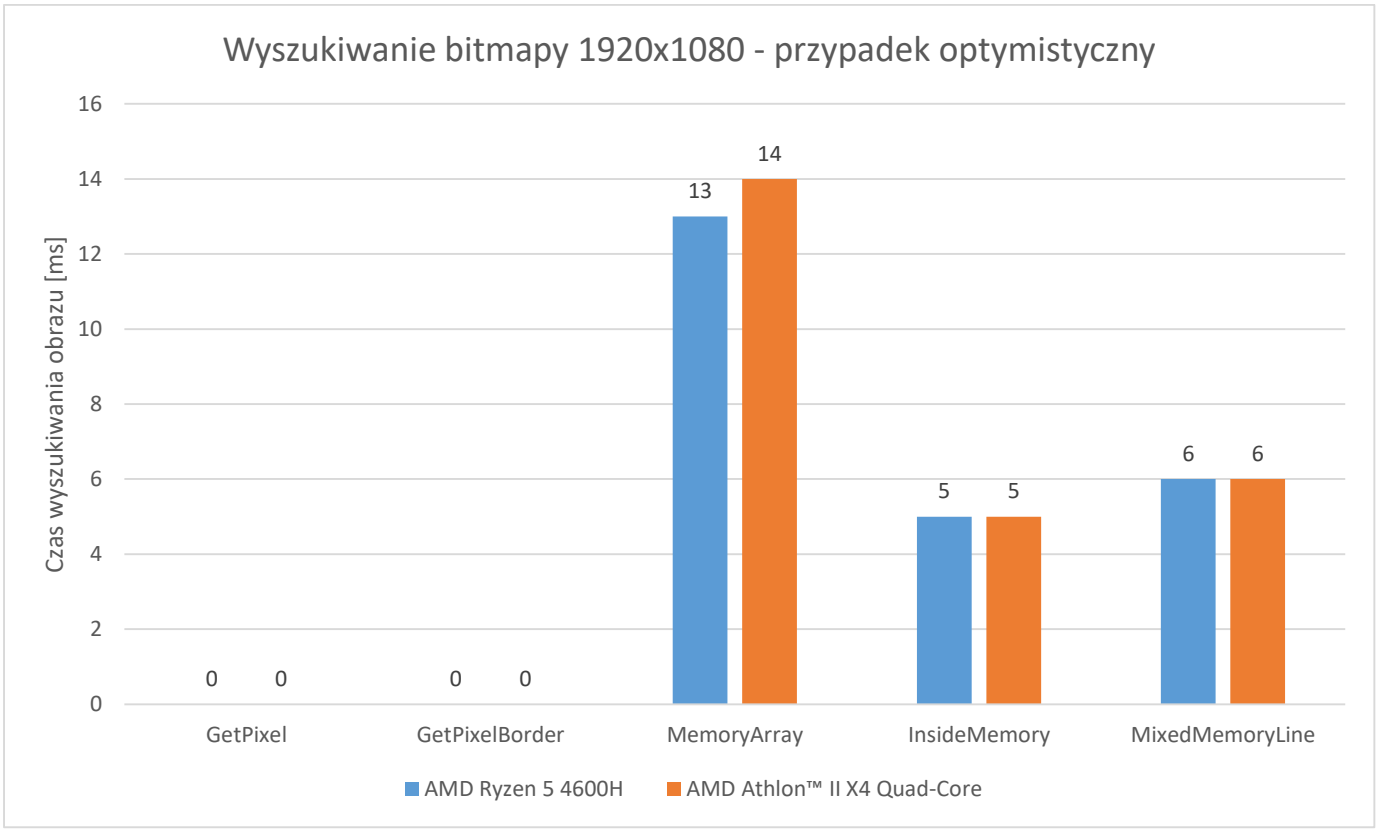
- MixedMemoryLine
  Połącznie algorytmów MemoryArray oraz InsideMemory pozwoliło uzyskać algorytm, który dla bitmapy poszukiwanej tworzy odpowiednik w postaci tablicy dwuwymiarowej wartości Intowych, a dla obszaru przeszukiwanego pracuje bezpośrednio na pamięci procesora.

Przypadek optymistyczny oraz pesymistyczny

W celu sprawdzenia możliwości granicznych algorytmów poddane one zostały badaniom z przypadkiem bitmapy optymistycznej oraz pesymistycznej. Od budowy algorytmu zależy jak będzie one reagować na napotkane przypadki przeszukiwania. Na rzecz tej analizy ustalony został wspólny przypadek optymistyczny oraz pesymistyczny dla wszystkich z nich.

Przypadek optymistyczny

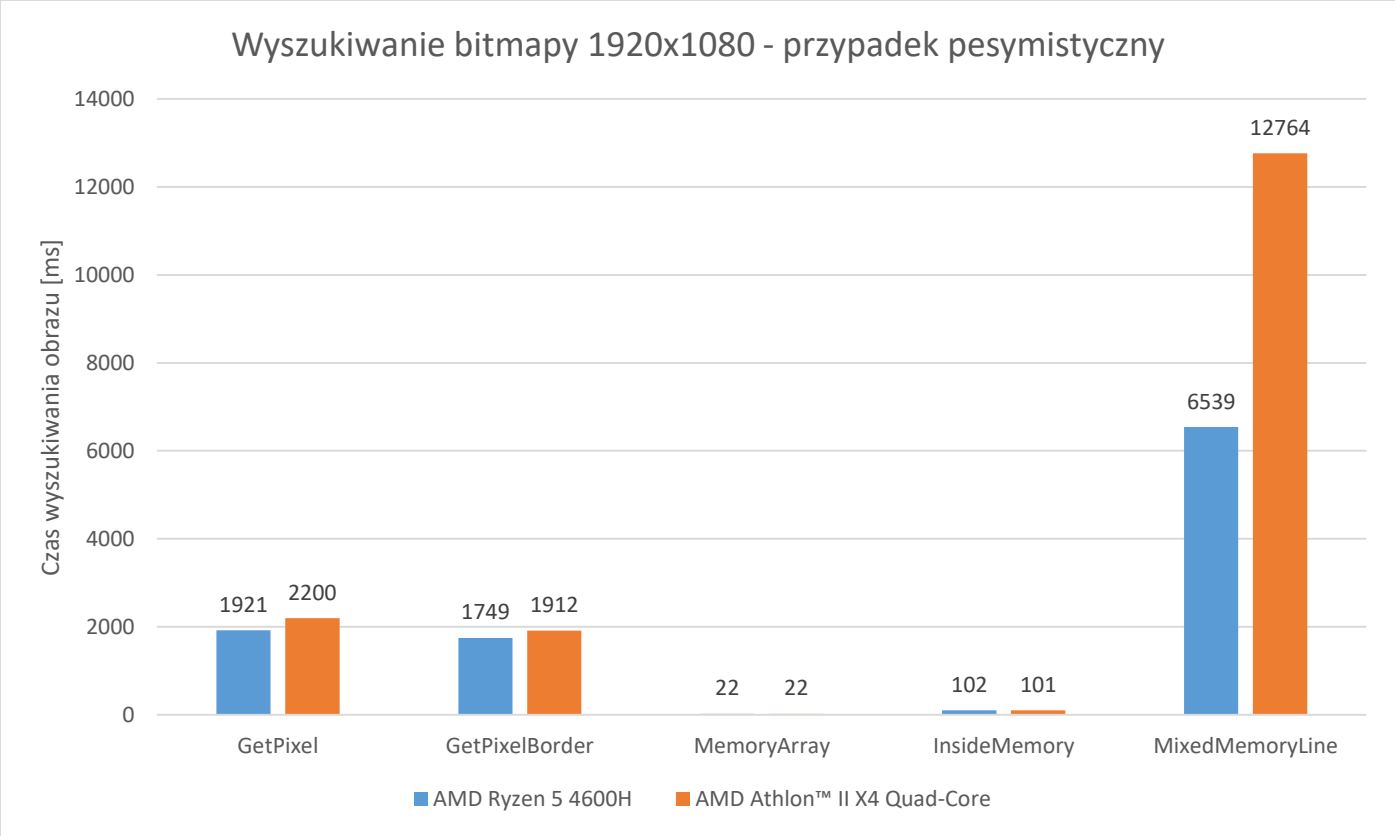
Przeszukiwania bitmapa ma jednolitą wartość pikseli na całej rozpiętości z wyjątkiem pierwszego piksela znajdującego się na koordynatach X: 0, Y: 0, który jest innej wartości.



Proste w budowie algorytmy wykonują na tyle szybko te wyszukiwanie, że na wykresie zajmuje im to 0 milisekund. Najdłuższą pracę wykonuje algorytm MemoryArray, który pomimo tak prostego zadania, tworzy przed każdym wyszukiwaniem tablicę wartości bitmapy co jak widać zajmuje mu w przybliżeniu ~ 13 – 14 milisekund.

Przypadek pesymistyczny

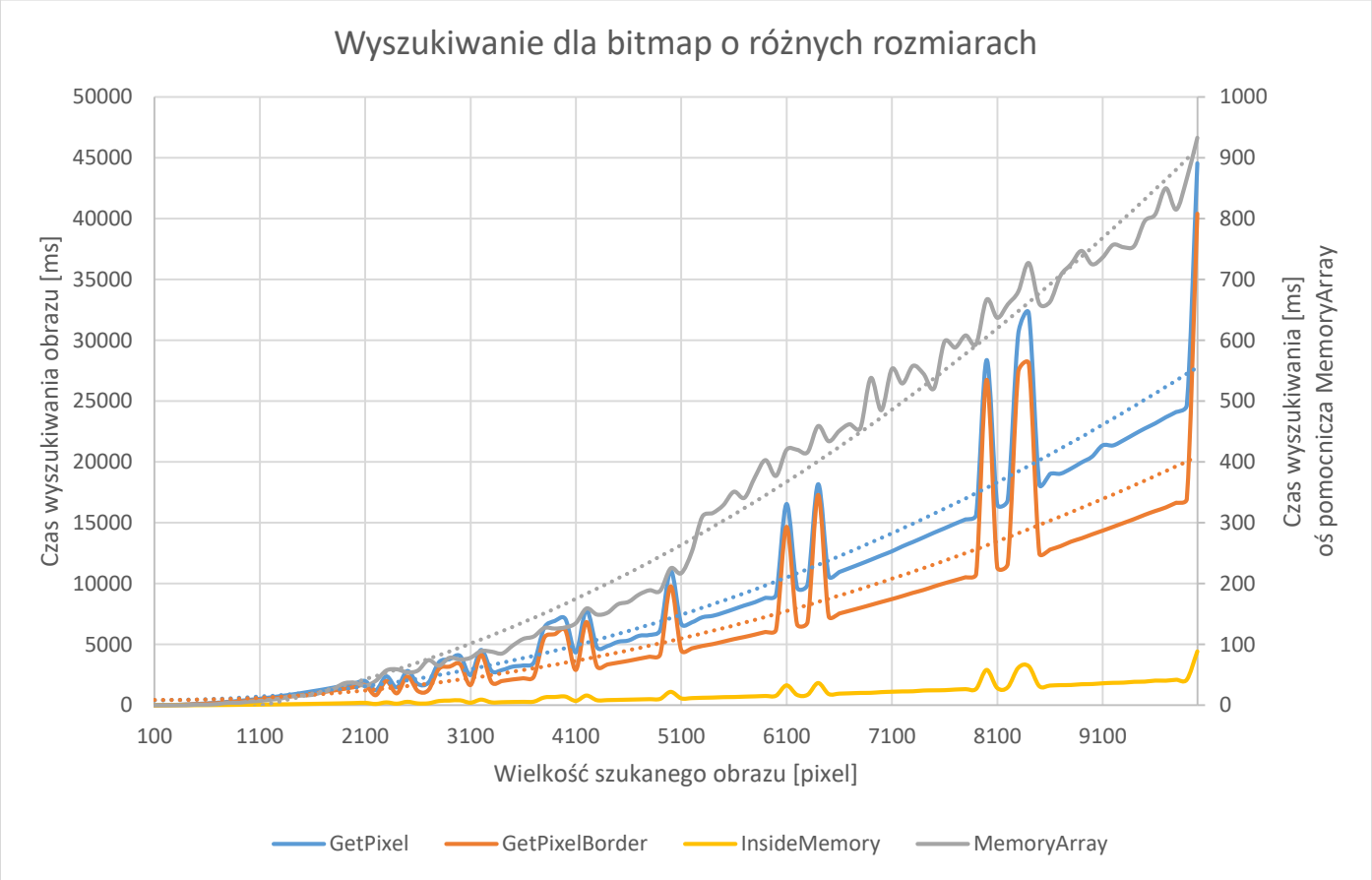
W przeciwieństwie do przypadku optymistycznego, szukana bitmapa to ¼ wielkości bitmapy bazowej na której algorytm dokonuje przeszukiwania. Obiekt szukany znajduje się na ostatniej ćwiartce głównej bitmapy czyli na koordynatach: X: 960, Y: 540. Dzięki takiemu rozstawieniu algorytmy muszą przeszukać większą część bazowej bitmapy.



Dla przypadku pesymistycznego widać już, że algorytm MixedMemoryLine jest niewydajny i odbiega zdecydowanie od reszty wyników. Algorytmy GetPixel oraz GetPixelBorder tym razem nie klasyfikują się na najwyższej pozycji, tym bardziej że różnica między pozostały jest naprawdę spora. Algorytmy MemoryArray oraz InsideMemory w tym przypadku są bardzo wydaje, natomiast badanie przeprowadzane jest na bitmapie o rozdzielczości tylko FullHD.

Rozmiar a wydajność

Przypadek optymistyczny oraz pesymistyczny, nie jest w stanie odzwierciedlić wszystkich przypadków, na które mogłyby natrafić algorytmy. Świetnym środowiskiem na zbadanie ich wydajności jest zasymulowanie dużej rozpiętości rozmiarowej przeszukiwanych bitmap. W tym doświadczeniu wszystkie algorytmy poddane zostały pomiarom czasowym rozpoczynając od bitmap wielkości 100, a kończąc na 1000x1000 pikseli. Punkty pomiarowe występowały przy zwiększeniu bitmapy o 100 pikseli wysokości oraz 100 pikseli szerokości, a wyniki były uśredniane z 10 prób pomiarowych dla każdej wielkości.



Wyniki na wykresie wydają się być nieregularne, natomiast jest to spowodowane zaburzeniami dostępu pamięci, w szczególności przy zmianie rozmiaru bitmap o dużych o dużych rozmiarach wahania są znaczące. Natomiast Podsumowaniem informacją, którą można wyciągnąć z tego wykresu jest to, że algorytm MemoryArray spisał się znakomicie przy tak dużych obiektach i zdeklasował pozostałem, dlatego został on przedstawiony za pomocą osi pomocniczej. Niemniej jednak algorytm InsideMemory, zasługuje również na uwagę ze względu na wciąż znacząco krótszy czas od pozostałych algorytmów. W tym doświadczeniu nie przetestowano algorytmu MixedMemoryLine, ponieważ jego czasy wyszukiwania był długie przy tak dużych obszarach przeszukiwania.

Wnioski

Bezprecedensowo najwydajniejszy okazuje się być algorytm MemoryArray, który tylko dla przypadku optymistycznego jednak klasuje się na ostatnim miejscu. Przypadek optymistyczny jednak jest na tyle rzadki do zaobserwowania, że jest dopuszczalny wynik, również należy zwrócić uwagę, że wielkości czasowe dla przypadku optymistycznego są niewielkie. Natomiast swoje możliwości algorytm ten pokazał dopiero dla badania opartego o różne rozmiary bitmapy, gdzie czasy wyszukiwania w porównaniu z resztą są drastycznie krótkie. Należałoby również wspomnieć o algorytmach GetPixel oraz GetPixelBorder, które okazały się zdecydowanie mało wydajne dla obszarów przeszukiwań o dużej ilości pikseli, jednak mimo to są proste w implementacji, mało awaryjne i trafne do zastosować o małej potrzebie wydajnościowej.